図解 VHDL実習

ゼロからわかるハードウェア記述言語

堀 桂太郎 著

森北出版株式会社

■著者紹介

堀　桂太郎（ほり・けいたろう）
明石工業高等専門学校助教授
博士（工学）

nto

_l) ;

写真提供：メメックジャパン株式会社
カバーデザイン：CELLENT JAPAN

図解 VHDL実習

——ゼロからわかるハードウェア記述言語——

堀 桂太郎 著

森北出版株式会社

まえがき

ディジタル回路の基礎を一通り学習された方は，シフトレジスタやカウンタなどを応用した回路を設計，製作できるようになっていることでしょう．この場合，従来では，図記号を用いた回路設計を行った後，必要な汎用ロジックICなどを用意してはんだごて片手に製作を行うという手順が一般的でした．しかし，現在ではHDL（ハードウェア記述言語）を用いることで，あたかもプログラミングを行うかのような感覚で設計を行い，設計データをCPLD/FPGAデバイスに転送するだけで実際のディジタル回路ができあがります．つまり，自分専用のICを1個からでも簡単に製作することができるようになったのです．簡単な組合せ回路からCPUなどの高機能なディジタル回路でさえCPLD/FPGAを用いて構成することが可能です．しかも，開発用ソフトウェアはフリーで入手することができます．この設計手法をマスターすることは，現代のディジタル回路設計技術者にとって不可欠な要素になっています．

本書は，はじめてHDLを学ぶ方々を対象に，広く普及しているVHDLを取り上げ，図を多く用いたわかりやすい解説を心がけました．そして，実際の開発ツールの操作手順を説明しながら，読者が実習を行いながら学習を進められるように配慮しました．扱う内容は，基本的な重要事項に絞り，欲張りすぎないように注意しました．しかし，本書の内容をマスターするだけでも，多くのディジタル回路をVHDLで自在に設計できるようになることでしょう．その後に，読者の皆様がさらに意欲をもって学習を進めて頂ければたいへん嬉しく思います．ただし，VHDLや各種の開発ツールは，決して万能ではありません．また，VHDLによる設計は，ソフトウェアではなく，ハードウェアを構成するための作業です．したがって，できあがる回路を常にイメージしながら設計を進めることが大切です．そのためには，ディジタル回路の基礎知識が土台になることを忘れないでください．本書が，VHDLマスターへ向けた最初の一歩を踏み出すための入門書としてお役に立つことを心より願っています．

最後になりましたが，本書執筆の機会を与えてくださった森北出版の森北博巳氏，ならびに編集でお世話になった石田昇司氏にこの場を借りて厚く御礼申し上げます．

2004年3月

著　者

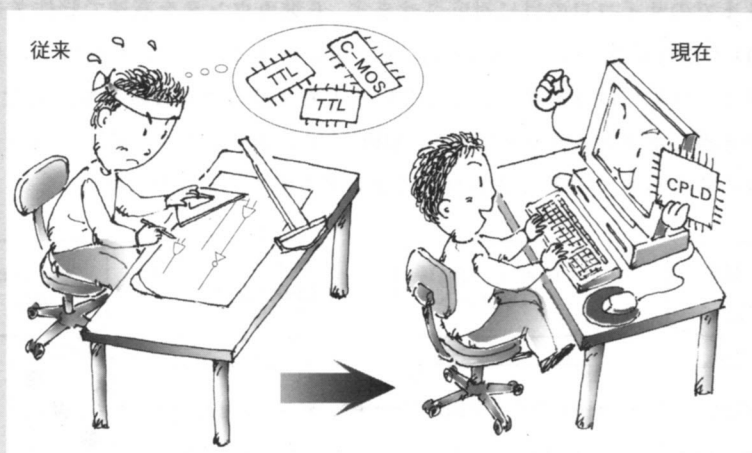
＜本書で扱ったソースファイルは，以下のHPから入手可能です＞
☆森北出版のHP : http://www.morikita.co.jp/soft/78391/
☆著者のHP : http://www.akashi.ac.jp/contents/Electric/hori/

目　次

*本書掲載の URL，画面写真等はとくに記述がない限り 2004 年 1 月時点のものです．
*HDL トレーナーについてのユーザサポートはソリトンウェーブ社で行います．その他のソフトウェアなどについては各社にお問い合わせください．

第1章 CPLD/FPGAの基礎

ディジタル回路を設計する場合，これまでは，設計者がAND，OR，NOTなどのゲート図記号を組合せて回路を設計していました．言い換えれば，TTLやC-MOSなどの汎用ロジックICを用いて回路を構成することが前提となっていたのです．しかし，現在ではCPLDやFPGAとよばれるデバイスを使用することを前提とし，各種の開発用ソフトウェアを用いた設計が主流になっています．

この章では，目的のディジタル回路を実現するためのいくつかの方法を説明したあと，CPLDやFPGAとはどのようなデバイスか，またこのデバイスを用いた回路設計の流れについて解説します．新しいディジタル回路設計手法を用いた場合のさまざまな利点や注意事項について理解しましょう．さらに，開発に必要なハードウェアやソフトウェアについての基礎を学びましょう．

1.1 ディジタル回路の設計方法

目的のディジタル回路を実現するためには，いくつかの方法があります．ここでは，汎用ロジックICを使用する方法，CPUを使用する方法，専用ICを使用する方法，CPLDやFPGAを使用する方法について説明します．

1.1.1 汎用ロジックICを使用する方法

たとえば，電子サイコロを製作する場合を考えましょう．図1.1に示すように，7個のLED（発光ダイオード）をサイコロの目に見立てて配置します．押しボタンスイッチを押した（ON）後に離す（OFF）と，LEDがサイコロの1～6までのいずれかの目をランダムに表示するようにします．

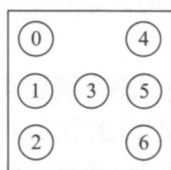



図1.1　LED表示部

ディジタル回路の基礎を学んだことのある方ならば，たとえば，図1.2に示すような回路構成を考えることができるでしょう．6進カウンタ回路の出力をデコーダ回路でデコードして，LEDの点灯を1～6までのサイコロの目に対応させます．6進カウンタを高速で動作させれば，人間の感覚ではLEDの点灯状態（サイコロの目）を予測できませんから，6進カウンタの動作を止めた時点でのサイコロの目はランダムになっていると考えることができます．したがって，6進カウンタの動作を押しボタンスイッチでON，OFFすればよいのです．押しボタンスイッチONで電子サイコロのスタート，OFFでストップ（ランダム表示）となります．

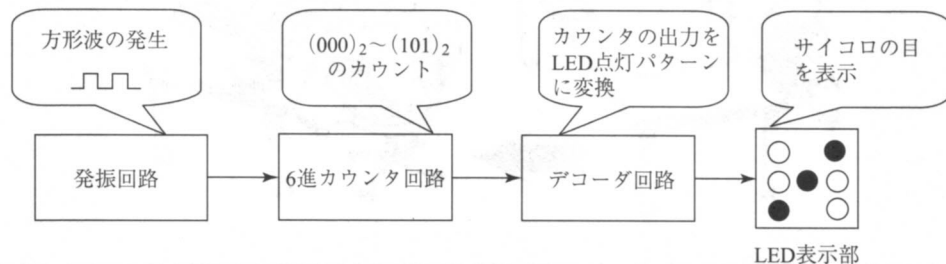



図1.2　電子サイコロの回路構成例

発振回路は，同期式6進カウンタを高速に動作させるための方形波を発生します．

もしも，電子サイコロの回路構成が理解できない場合でも，雰囲気だけをつかんでもらえば結構です．

この回路を従来の方法で設計する場合には，図1.3に示すように，6進カウンタ回路とデコーダ回路の動作を真理値表で表し，そこから得た論理式をカルノー図やクワイン・マクラスキー法

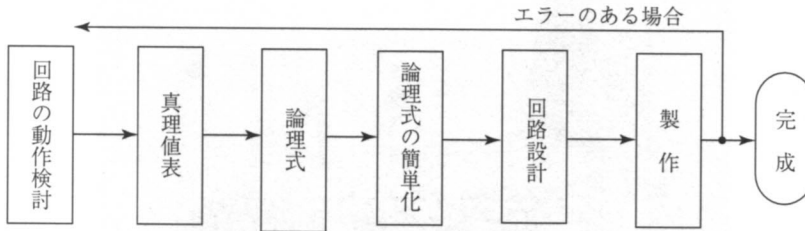



図1.3　従来のディジタル回路設計の流れ

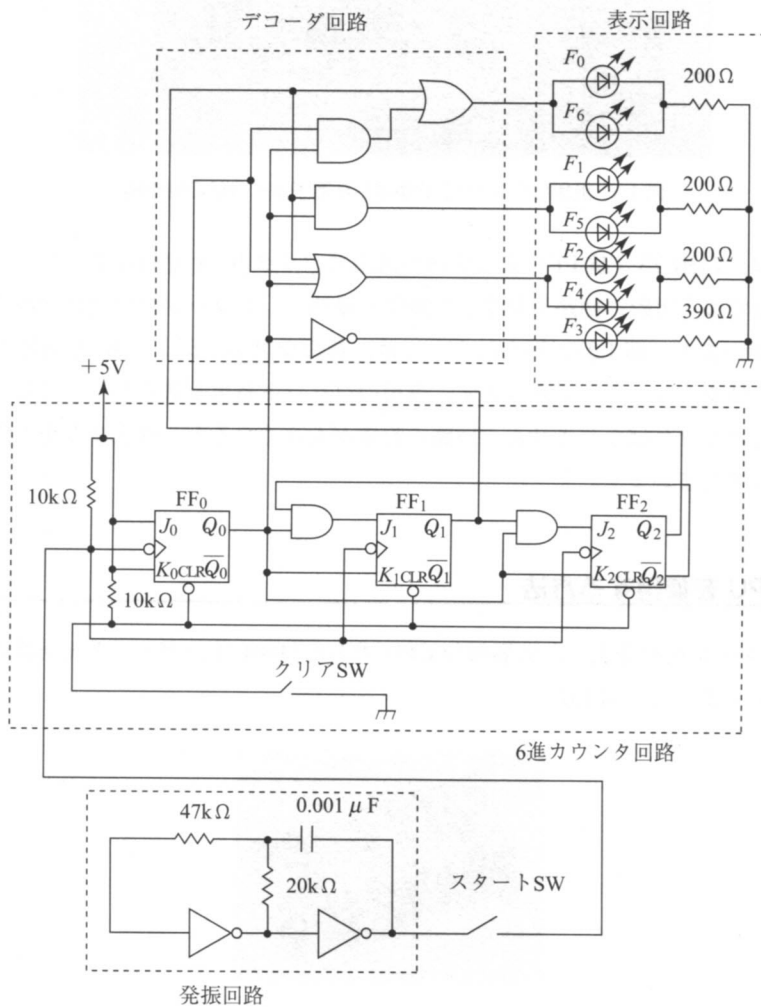



図1.4　汎用ロジックICを使用した電子サイコロ回路の例

などによって簡単化します．そして，AND，OR，NOT，フリップフロップ（FF）などの汎用ロジックICを用いて回路を設計するのが一般的でした．

このような手順によって設計した回路例を図1.4，実際の製作例を図1.5に示します．

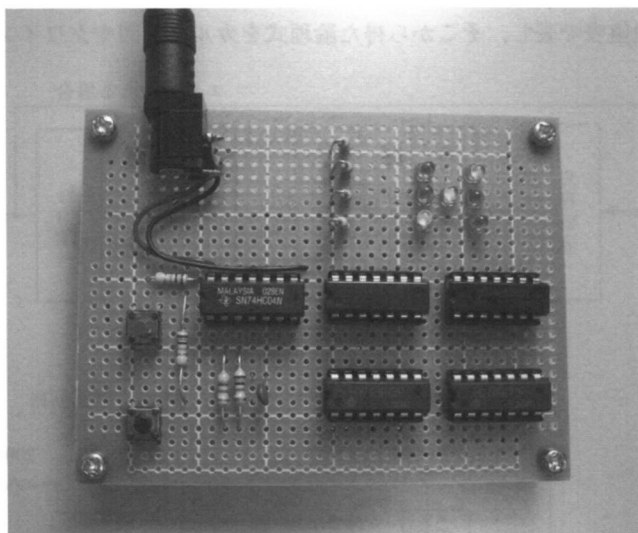



図1.5　汎用ロジックICを使用した電子サイコロの製作例

この回路では，74HC73（*JK*-FF）など5個の汎用ロジックICを使用しました．このような従来法による設計では，実際の回路を製作して動作を確認し，エラーが見つかれば前の段階に戻って検証作業を行います（図1.3）．場合によっては，新たな汎用ロジックICを追加して回路を製作し直さなければならないこともあります．専用のプリント基板を製作している場合には，基板の作り直しも行わなければなりません．回路の規模が大きくなると，修正や変更の作業はより困難なものとなってしまいます．

1.1.2　CPUを使用する方法

1971年にインテル社の発表した世界初のCPUであるi4004は，ディジタル回路の世界に大きな革命をもたらしました（図1.6）．

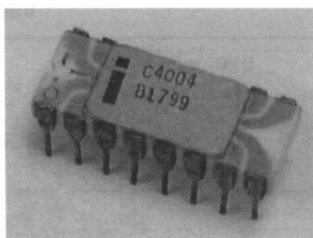



図1.6　i4004（インテル株式会社）

CPUが発表されるまでは，ICを用途ごとに設計して製作していたため，一種類のICを作るだけでも膨大な時間と費用が必要でした．しかし，CPUを使用すれば，ソフトウェア（プログラム）を変更することで目的の動作を実現することができるようになったのです．i4004の発表以降，CPUは汎用性の高い万能ICとして発展を続けています．

前に紹介した電子サイコロ回路の動作は，CPUを使って実現することもできます．たとえば，安価で扱いの簡単なPICマイコンを例に挙げましょう．PICマイコンは，マイクロチップテクノロジー社の開発した高性能マイコンで，1個のパッケージにCPUやメモリ機能などを内蔵しており，制御用として広く利用されています．図1.7に，PICマイコンの外観例を示します．

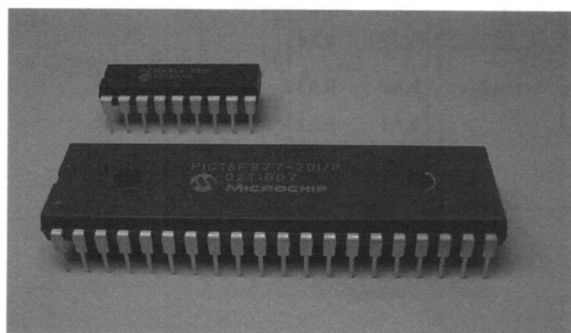



図1.7　PICマイコンの外観例（上がPIC16F84A）

PICマイコンを使用して電子サイコロを製作する場合には，入出力ピンに押しボタンスイッチやLEDを接続して，ランダムなサイコロの目を表示するようなプログラムを与えます．図1.8に，CPU（マイコン）を用いた設計の流れを示します．この場合には，6進カウンタやデコーダといったディジタル回路を意識する必要はありません．LEDの表示を高速で1～6まで繰り返し変化させ，押しボタンスイッチを離したときに，表示が停止するようにプログラムを作成すればよいのです．

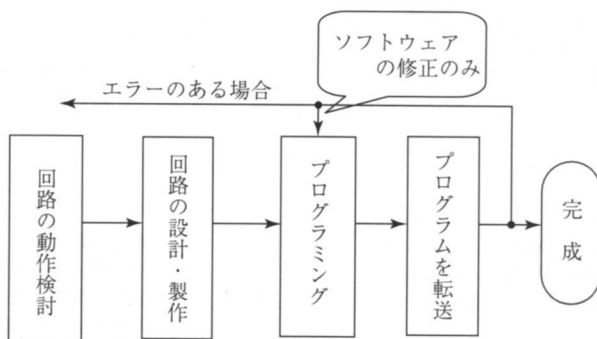



図1.8　CPU（マイコン）を用いた設計の流れ

図1.9にPICマイコン（PIC16F84A）を使用した電子サイコロの回路，リスト1.1にC言語によるプログラム例を示します．

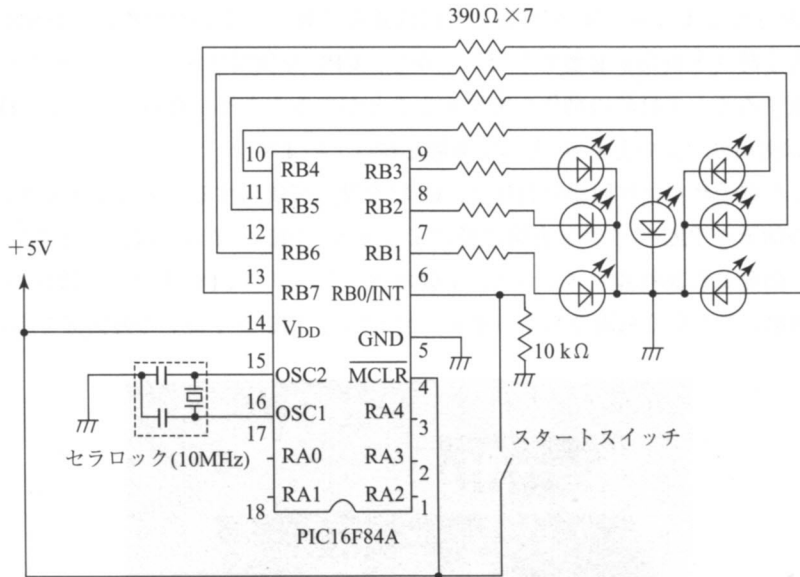



図 1.9　PIC マイコンを使用した電子サイコロ回路の例

```
// リスト1
// PIC による電子サイコロのプログラム

#include        <16F84A.h>
#use            delay(clock=10000000)
#byte           port_b = 0x06
int i ;                                                     // グローバル変数の宣言
#int_ext
int_isr(){                                                  // 外部割込み関数
int eye[ 6] = { 0x10, 0x22, 0x32, 0xAA, 0xBA, 0xEE } ;     // サイコロの目データ
        port_b = eye[ i] ;                                  // ポート B にデータ出力
        delay_ms(1000) ;                                    // 1 秒タイマの呼出し
        port_b = 0x00 ;                                     // ポート B クリア
}

main()
{
        enable_interrupts(global);                  // 割込み許可
        enable_interrupts(int_ext) ;                // 外部割込み許可
        ext_int_edge(L_TO_H) ;                      // 割込み信号は立上りエッジ
        set_tris_b(0) ;                             // ポート B を出力に設定

        while(1)                                    // 割込み待ちループ
                for(i = 0 ; i <= 5 ; i++)           // 乱数発生ループ
                        ;
}
```

リスト 1.1　電子サイコロのプログラム例（CCS コンパイラ）

図1.10に，実際の製作例を示します．同じような動作をする回路であるにもかかわらず，汎用ロジックICを使用した場合（図1.5）と比べると，ICはわずか1個で済んでいます．また，動作の仕方を変更する場合などは，回路はそのままで，プログラムだけを変更することで対応できる利点があります．ここで，CPUに与えたプログラムは，ハードウェア上で動作するソフトウェアであるという点に注目してください．あとで説明するCPLDやFPGAでは，プログラムによってハードウェアの構築を行うという点が異なります．また，CPUを用いた場合には，CPUがプログラムを解釈しながら実行をするために，ハードウェアだけで構成したディジタル回路のようには高速動作をさせることはできません．

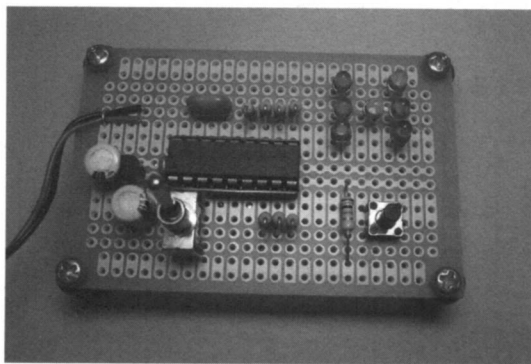



図1.10　CPU（PICマイコン）を使用した電子サイコロの製作例

▶1.1.3　専用ICを使用する方法

前に説明した汎用ロジックICを使用して電子サイコロを製作する場合において，図1.11に示すように，6進カウンタ回路とデコーダ回路の部分を専用ICとして製作する方法があります．発振回路は，アナログ回路ととらえてIC化から除外しています．この例のように，ある用途のために作られた専用ICのことを，ASIC（application specific IC：特定の用途向けICという意味），またはカスタムICといいます．ここでは，簡単な電子サイコロの回路を例に挙げていますが，実際の工業製品では非常に複雑な回路を扱う場合が多くあります．

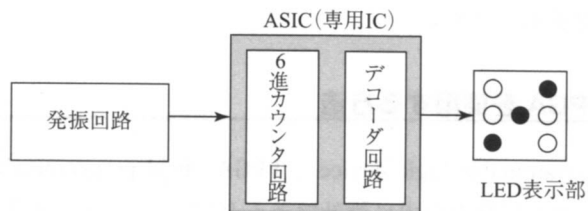



図1.11　ASICを使用した電子サイコロ回路の構成

ASICを用いれば製品の製作は簡単になり，高速な動作を実現することもできます．しかし，ASICを作るには，専門のICメーカに依頼する必要があり，ある程度の量産（最低でも5000～10000個程度）が条件になります．総費用は非常に高くなり，できあがるまでに時間がかかるという問題も生じます．また，ASICが本当に正しく動作するかどうかは，実際にできあがった製品で試してみるまでわかりません．仮に，テスト段階で動作を確認したとしても，それをIC化した場合には予期せぬトラブルが発生することもあります．もしも，できあがったASICが正常に動作しなかった場合には，膨大な損失を被ることになってしまいます．一説では，完全に正常動作するASICができあがる実際の成功率は，80％程度ともいわれています．全く使い物にならない場合は10～20％，ASIC外部に補正回路を接続することで何とか活かす場合も同程度あるようです．

図1.12に，ASICを製作するまでの大まかな流れを示します．

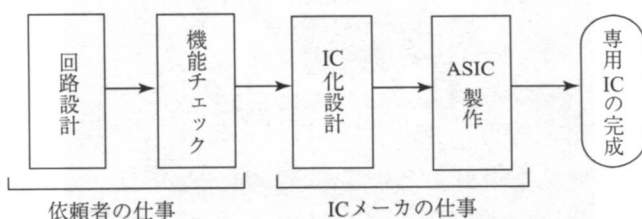



図1.12　ASIC製作までの流れ

つぎに，ASICの長所と短所をまとめておきます．

＜ASICの長所＞

・高速動作を実現できる

・省スペース化を実現できる（必要最小限のサイズでICを作ることができる）

・量産すれば，1個あたりの価格は安くなる

＜ASICの短所＞

・正常に動作しなかった場合のリスクが大きい

・できあがるまでに時間がかかる

・量産が条件となる

・後からIC内部の回路変更ができない

▶1.1.4　CPLD/FPGAを使用する方法

CPLD（complex programmable logic device），FPGA（field programmable gate array）とは，ユーザが内部のディジタル回路を自由に設計できるICのことです．本書では両者をまとめてCPLD/FPGAと表現しています．CPLD/FPGAについては，後で詳しく説明しますので，ここでは，これらのICを使った回路設計の概要をみてみましょう．

専用ICであるASICは，ICメーカに製作を依頼する必要がありました．しかし，ユーザ自身

がCPLD/FPGAとよばれる汎用ICを専用ICに仕立て上げる方法があります．これまでと同様に，電子サイコロを製作する場合を例に挙げて考えましょう．図1.13に示すように，6進カウンタ回路とデコーダ回路の部分をCPLD/FPGAで実現します．この構成については，ASIC（図1.11）の場合と同様です．

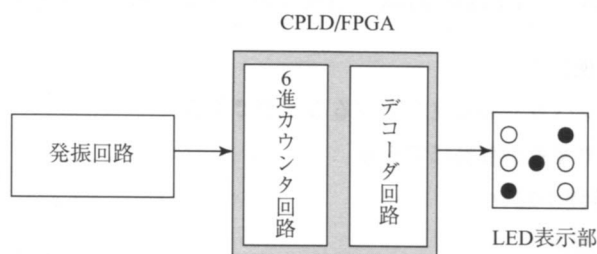



図1.13　CPLD/FPGAを使用した電子サイコロ回路の構成

CPLD/FPGAは，内部にたくさんのゲート回路やそれらの配線用機能などを備えており，ユーザが目的に応じた回路に仕立て上げることができるようになっています．いろいろな規格のCPLD/FPGAが販売されていますので，ユーザは目的のディジタル回路が実現できるCPLD/FPGAを選択して購入します．そして，内部の配線などを指示するデータをCPLD/FPGAに転送すればよいのです．

図1.14に，CPLD/FPGAを目的のディジタル回路に仕立て上げるまでの大まかな流れを示します．

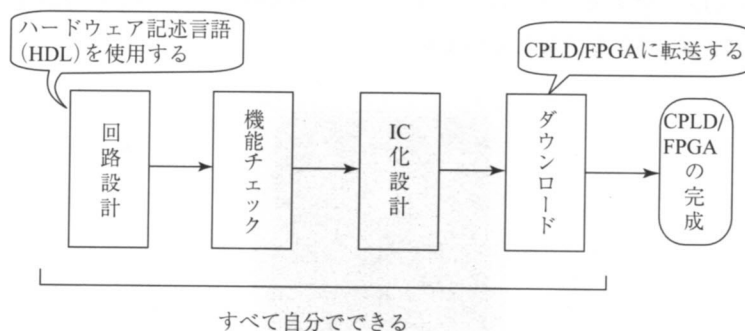



図1.14　CPLD/FPGAを仕立て上げるまでの流れ

図1.14における回路設計の作業は，従来のように汎用ロジックICを使った回路図を書き上げるものではありません．HDL（hardware description language）とよばれるディジタル回路の構成を表現するハードウェア記述言語を使用します．これは，一見C言語によるプログラムなどと似ています．しかし，C言語ではCPUに与える動作手順を記述するのに対して，HDLではCPLD/FPGA内部を目的のディジタル回路に仕立て上げるコードを記述する点が異なります．HDLで記述されたコードは，開発用ソフトウェアによってCPLD/FPGA内部で構成するディジタル回路を指示するデータに自動変換されます．このデータを転送（ダウンロード）すれば，CPLD/FPGAは内部の配線を変更することでユーザ専用ICとして動作するのです．

前に説明したASICを製作する流れ（図1.12）の「回路設計」においても，従来は汎用ロジックICを用いた回路設計法が用いられていましたが，現在ではHDLと各種の開発用ソフトウェアを使用した設計が主流になっています．HDLには，VHDLとよばれるものとVerilog-HDLとよばれるものが普及していますが，本書では，VHDLを中心にした説明を行います．

つぎに，ASICと比較したCPLD/FPGAの長所と短所をまとめておきます．

＜CPLD/FPGAの長所＞

・1個のICから個人レベルでも仕立て上げることができる
・IC内部の回路修正や変更が容易にできる
・できあがるまでの時間が短い

＜CPLD/FPGAの短所＞

・大量生産する場合には，ASICより1個あたりの価格が高くなる
・ASICほどは高速に動作しない
・スペースについてASICほどの最適化はできない

いくつかの短所がありますが，用途によって，特に小規模な開発で使用する場合には，それほど大きな問題とはならないものばかりでしょう．また，近年では，CPLD/FPGAの高性能化と低価格化が進み，ASICに取って代わることも多くなってきました．特に，製作までの時間が短く，回路修正が容易であるために，製品を早期に市場に投入できることはCPLD/FPGAを使う大きな利点となっています．

たとえば，図1.15に示すデバイスは，ザイリンクス社が販売しているCPLDで，1600個程度のゲート回路と72個のFFを内蔵しています．

図1.15　CPLDの例（ザイリンクス社XC9572）

リスト1.2は，電子サイコロ回路を実現するためのハードウェアをVHDLで記述したソースコードです．このソースコードから生成するデータをCPLDに転送すれば，CPLDは電子サイコロ回路として動作する専用ICへと変身するのです．

＜CPLD/FPGAの特徴＞

CPUは，ソースプログラム（C言語）で書かれた命令をハードウェア上で実行します．

CPLD/FPGAは，ソースコード（HDL）で書かれたハードウェアを構築します．

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity dice is
   Port ( LED : out std_logic_vector(6 downto 0);
          CLK : in std_logic;
          ST : in std_logic);
end dice;

architecture Behavional of dice is

signal CNT: std_logic_vector(2 downto 0);
signal Q : std_logic_vector(15 downto 0);
signal CE : std_logic;

begin

process (CLK)
begin

   if CLK'event and CLK='1' then
        Q <= Q + '1';
   end if;
end process;

process (Q)
begin
   if Q = "1111111111111111" then
      CE <= '1';
   else
      CE <= '0';
   end if;
end process;

process(CE)
   begin
      if(CE'event and CE='1') then
              if(ST = '1') then
                if(CNT = "101") then
                      CNT <= "000";
                else
                      CNT <= CNT + '1';
              end if;
              end if;
         end if;
   end process;

process(CNT)
    begin
          case CNT is
                 when "000" =>
                         LED <= "0001000";
                 when "001" =>
                         LED <= "0010001";
                 when "010" =>
                         LED <= "0011001";
                 when "011" =>
                         LED <= "1010101";
                 when "100" =>
                         LED <= "1011101";
                 when others =>
                         LED <= "1110111";
          end case;
end process;
end Behavional;
```

注：スイッチで生じるチャタリングの影響を避けるために，クロック分周回路を入れてあります．

リスト1.2　電子サイコロの記述例（VHDL）

図1.16　CPLDを使用した電子サイコロ回路の例

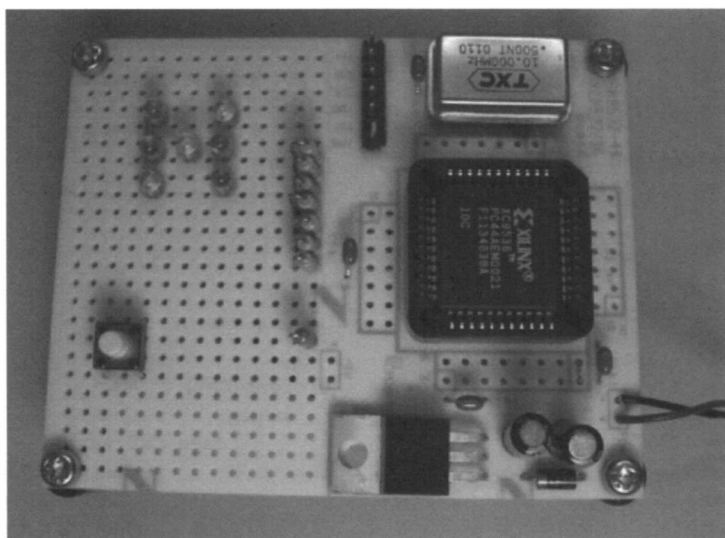



図1.17　CPLDを使用した電子サイコロの製作例

図1.16にCPLDを用いた電子サイコロの回路，図1.17に実際の製作例を示します．

CPLD/FPGAを使用すれば，それらをCPUに仕立て上げることさえ可能です．そうすれば，使いたいCPUが製造中止になることを心配する必要などはありません．

本書の目的は，VHDLを使用してCPLD/FPGAを使いこなせるようになるための基礎を身につけることです．

1.2 CPLD/FPGA

ユーザが目的に応じて自由にカスタマイズできるデバイスは,CPLD と FPGA に大別できます. 初級レベルの VHDL を記述するならば, どちらのデバイスを使用する場合であっても記述内容に変わりはありません. しかし, 複雑で厳密な動作が要求されるディジタル回路を構成する場合には, 使用するデバイスのハードウェアを考慮した記述が要求されることもあります. ここでは, CPLD, FPGA それぞれの特徴や差異についての基礎を学びましょう.

1.2.1 CPLD

CPLD (complex programmable logic device) は, プログラム可能な大規模論理素子という意味をもつデバイスの略称です. 図 1.18 に, CPLD の基本構造の例を示します.

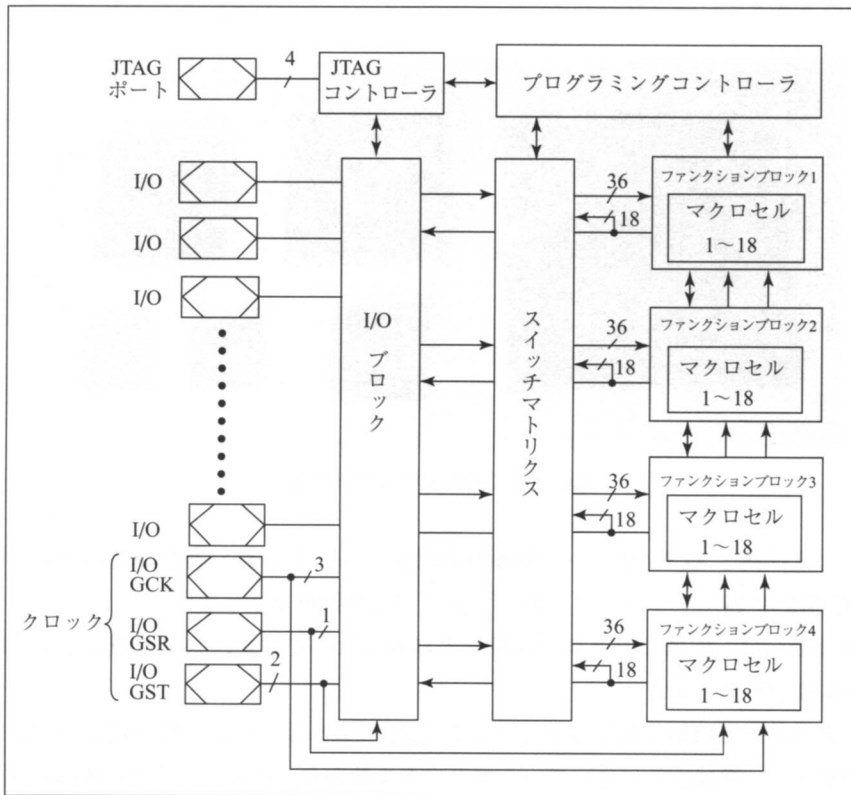



図 1.18 CPLD の基本構造の例

CPLDは，機種によって異なる数のファンクションブロック（function block）を有しており，ファンクションブロック内には組合せ回路とマクロセル（macrocell）とよばれるフリップフロップが備わっています．図1.18の例では，4個のファンクションブロック内に，おのおの18個のマクロセルが格納されています．スイッチマトリクス（switch matrix）は，格子状に配置された多数の配線に対してワイヤードAND機能を用いて目的の回路を形成するように接続する働きをします．また，I/Oブロック（I/O block）は，入出力ピンを制御しています．JTAG（joint test action group）コントローラとプログラミングコントローラは，JTAGとよばれるプロトコル（通信規約）でデータ書込みを実現する機能です．

CPLDの規模を選定する場合には，マクロセルの数をひとつの目安にするとよいでしょう．CPLDは，フラッシュメモリ（EEPROM）の技術を用いて作られていますので，データを転送した後に電源を切っても，データを保持することが可能です．つまり，カスタムICとしての構成を保持できます．データは，再書込み可能で，およそ1万回以上の消去回数が保証されています．CPLDは，小/中規模な回路を実現する場合に適したデバイスです．

ザイリンクス社のCPLDを例に挙げると，XC9500，CoolRunner，CoolRunner-IIなどのシリーズがあります．新しいシリーズになるにしたがって，高速化と低消費電力化が進んでいます．これらのCPLDは，同じ機種でも異なるパッケージ（外形）がありますので，用途に応じた選択をすることができます．図1.19に，CPLDのいくつかのパッケージ例を示します．

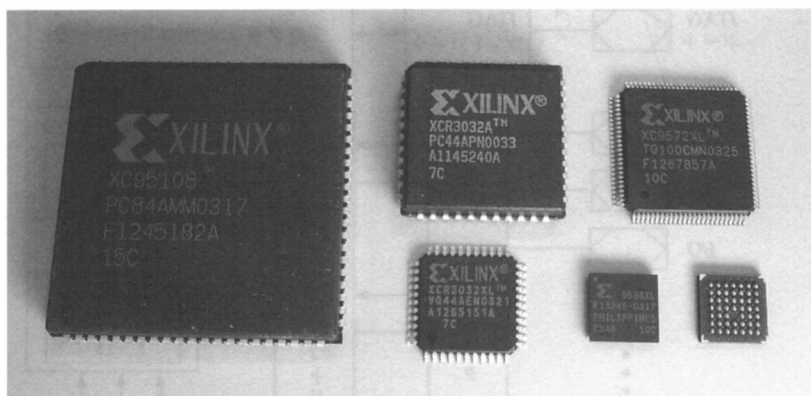



図1.19　CPLDのパッケージ例

また，たとえば同じXC9572という型番のCPLDでも，ピン数の異なる製品があります．図1.20に，XC9572の44ピンタイプと，84ピンタイプの外観例を示します．

このような製品は，内部のマクロセル数などは同じなのですが，入出力用として使用できるI/Oピンの本数が異なります．

ザイリンクス社のCPLDを例に挙げると，図1.21のようなルールで機種番号が付けられています．したがって，機種番号に付いているマクロセル数から，そのデバイスの規模を類推することが可能です．

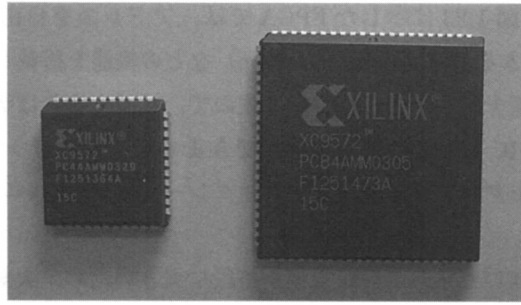



図 1.20 ピン数の異なる CPLD（どちらも XC9572）

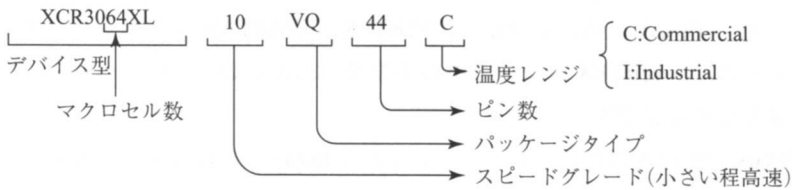



図 1.21 CPLD の機種番号の例（ザイリンクス社）

1.2.2 FPGA

FPGA（field programmable gate array）は，現場でプログラム可能な論理素子という意味をもつデバイスの略称です．図 1.22 に，FPGA の基本構造の例を示します．

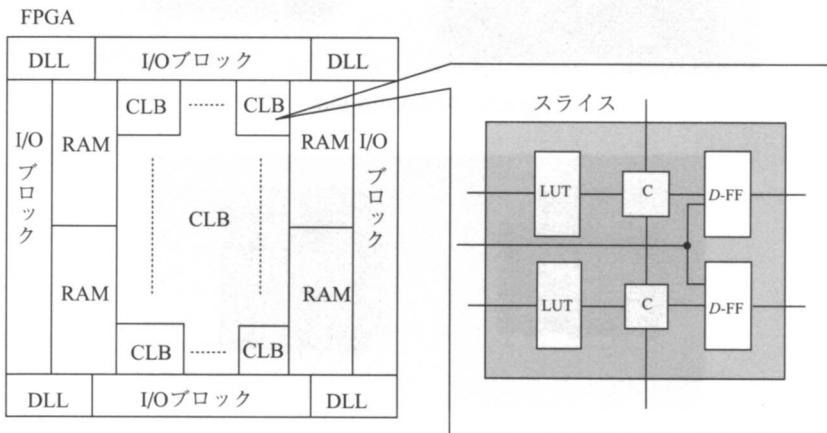



図 1.22 FPGA の基本構造の例

FPGA には，多数の CLB（configurable logic block）とよばれるブロックが敷き詰められており，各 CLB 内には数個程度のスライス（slice）とよばれる機能単位が内蔵されています．各スライスには，2 個の *D*-FF と C（carry）とよばれる演算回路および，LUT（look up table）とよばれるプログラム可能な組合せ回路が備わっています．前に説明した CPLD ではファンクションブロックという基本機能を備えていましたが，FPGA ではそれにあたる機能が CLB というこ

とになります．さらに，図1.23に示したFPGAでは，アドレスを自由に設定して使用できるRAM，クロックを制御できるDLL（delay look loop）などの機能を搭載しています．

FPGAは，SRAMの技術を用いて作られていますので，電源を切ればデータは消失してしまいます．つまり，カスタムICとしての構成を保持できません．したがって，電源投入のたびに，外部のROMからデータをダウンロード（コンフィグレーションともいいます）することが必要となります．

しかしFPGAは，CPLDにくらべて回路規模が格段に大きいため，大規模な回路を実現する場合に適したデバイスです．たとえば，CPLDに使用しているゲート数の総和はおよそ750～12,000個程度ですが，FPGAではおよそ5,000～8,000,000個にまでおよびます．

ザイリンクス社のFPGAを例に挙げると，VIRTEX，SPARTANなどのシリーズがあります．SPARTANシリーズは，VIRTEXシリーズ中の小規模（およそ20万ゲート以下）のデバイスを低価格化した量産型の製品です．

図1.23にFPGA，図1.24にコンフィグレーション用ROMの外観例を示します．

図1.23　FPGAの外観例

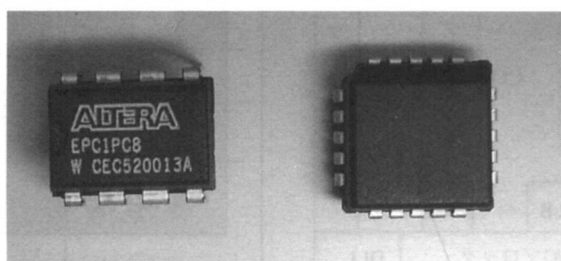



図1.24　コンフィグレーション用ROMの外観例

1.3 ハードウェア記述言語

CPLD/FPGAの内部構造を決めるためには，ハードウェア記述言語（HDL）を使用してソースコードを書くことが必要です．HDLには，いくつかの種類があります．ここでは，HDLの概要とHDLを用いた設計手順などについての基礎を学びましょう．

1.3.1 ハードウェア記述言語の種類

代表的なHDLには，つぎのようなものがあります．

（1）VHDL

1980年代にアメリカ国防総省の超高速IC（VHSIC : very high speed integrated circuit）開発プロジェクトによってまとめられたHDLです．VHDLの頭文字は，このプロジェクト名のVに由来しています．1987年，IEEEによって標準化されて以来，バージョンアップを続けている代表的なHDLです．

（2）Verilog-HDL

アメリカのゲートウェイ・デザイン・オートメーション社（後にケイデンス社と合併）によって開発されたHDLです．当初は，論理シミュレーション用の言語として登場しました．IEEEによる標準化は，VHDLよりも遅い1995年です．

（3）AHDL

CPLD/FPGA開発メーカであるアメリカのアルテラ社が開発したHDLです．

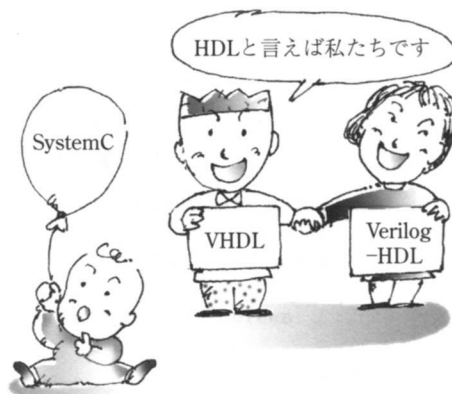



図1.25　VHDLとVerilog-HDLが主流

(4) ABEL

アメリカのデータ I/O 社の開発した HDL です．

(5) SFL

日本の NTT が開発した PARTHENON というハードウェア開発環境で使用するための HDL です．SFL は structured functional language の略称です．

(6) systemC

OSCI（open systemC initiative）によって 2000 年に発表された設計言語です．C++ を基本としているために，すでに C++ をマスターしている技術者にとっては修得しやすいといわれています．単にハードウェアを記述するだけでなく，そのハードウェア上で動作するソフトウェアの記述もできるシステムレベル設計言語とよばれる仕様をしています．

現在の産業界では，VHDL と Verilog-HDL が HDL の主流となっています．新しく登場してきた systemC や specC などは，大いに注目されてはいますが，いまだ発展途上といっていいでしょう．また，VHDL と Verilog-HDL を比較すると，VHDL のほうが厳格な記述ルールをもっています．しかし，どちらの HDL を使用したとしても，CPLD/FPGA や開発ツールが変わるわけではなく，同様の手順で設計を行うことができます．リスト 1.3 に VHDL と Verilog-HDL の簡単な記述例を示します．

本書では，HDL の中で最も歴史のある VHDL を中心にしてハードウェアを記述する方法を解説します．

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity LED_VHDL is
    Port ( SW : in std_logic;
           LED : out std_logic_vector(6 downto 0));
end LED_VHDL;

architecture Behavioral of LED_VHDL is

begin

process(SW)
begin
 if SW='0' then
   LED <= "0000000";
 else
   LED <= "1111111";
 end if;
end process;
end Behavioral;
```

(a) VHDL

```
module LED_VERILOG(SW,LED);
    input SW;
    output [ 6:0] LED;

        assign LED=(SW==1'b1)?7'b1111111:7'b0000000;

endmodule
```

(b) Verilog-HDL

リスト 1.3　HDL の記述例

▶1.3.2　ハードウェア記述言語を用いた設計手順

図 1.26 に，HDL を用いた設計の流れを示します．この流れは，VHDL，Verilog-HDL とも同じです．

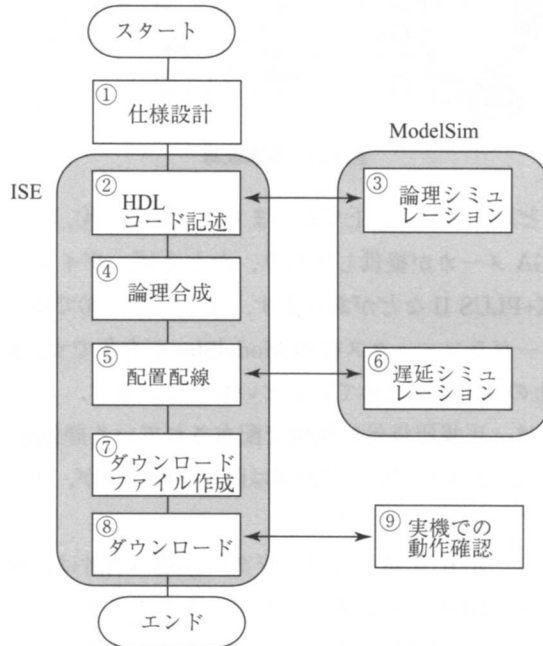



図 1.26　HDL を用いた設計の流れ

① 仕様設計：どのような回路を製作したいのかを明確にします．

② HDL コード記述：VHDL または Verilog-HDL でコードを記述します（リスト 1.3 参照）．コードは，テキストファイル形式で保存します．

③ 論理シミュレーション：②で記述したコードを用いて，ハードウェアの動作をシミュレーションして検証します．ここでのシミュレーションは，実際の回路で生じる遅延などを考慮しないため，論理シミュレーションとよばれます．小規模な回路では，シミュレーションを省略することもあります．

④ 論理合成：論理的な動作を確認したコードをもとに，ディジタル回路を設計します．

⑤ 配置配線：④で設計したディジタル回路を実際に使用する CPLD/FPGA 内部で構成するためのデータを生成し，ピンの割り当てを行います（図 1.27 参照）．

⑥ 遅延シミュレーション：⑤の配置配線によって，使用する CPLD/FPGA のピン割り当てなどが決まったので，遅延を含めたシミュレーションを行います．

⑦ ダウンロードファイル作成：CPLD/FPGA へ転送できる形式のファイルを作成します．

⑧ ダウンロード：CPLD/FPGA とパソコンをケーブルで接続してファイルの転送を行います．ここでは，ファイルを CPLD/FPGA に転送することをダウンロードとよんでいます．

⑨ 実機での動作確認：CPLD/FPGA を実機に実装して動作確認をします．

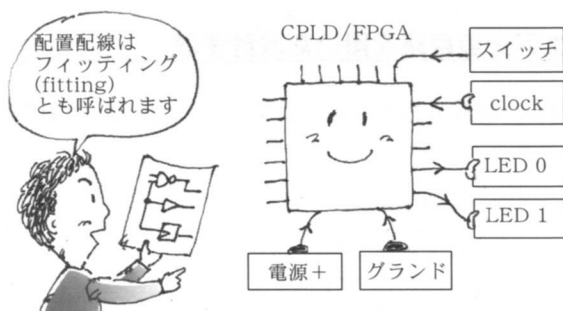



図1.27　配置配線

これらの作業のほとんどは，パソコン上で行います．上記②，④，⑤，⑦，⑧で使用するソフトウェアは，CPLD/FPGAメーカが提供しており，たとえば，ザイリンクス社のISEやアルテラ社のQuartus II，MAX+PLUS IIなどがあります．また，③，⑥で使用するシミュレーションソフトウェアは，メンターグラフィックス社のModelSimが有名です．ModelSimは，この分野で世界のおよそ70％以上のシェアを占めているといわれています．

これらのソフトウェアは，正規版以外に無償で配布されている評価版やWeb（ウェブ）版などがあります．ウェブ版は，正規版に比べて機能は限定されますが，大規模な開発以外なら十分に使用できる性能をもっています．

これまで説明したように，HDLを用いたディジタル回路の設計は，回路図を用いての設計とはだいぶ様子が異なります．HDLでは，あたかもソフトウェアを開発するかのような雰囲気で回路設計を行うのです．しかし，HDLによって回路を設計する場合でも，ディジタル回路に関するハードウェアの知識は不可欠です．たとえば，実際のディジタル回路では，伝達遅延時間が必ず生じます．回路の規模が大きくなると，この伝達遅延時間を十分に考慮した設計をしなければ正しい動作は期待できません．したがって，HDLを用いた設計においては，出来上がる回路の様子や規模をイメージしながら作業を行うことが重要になります．また，残念ながらHLDや関連する開発ツールは完全に万能なものではありません．よりよい回路を設計するためにも，ディジタル回路の知識が必要になります．

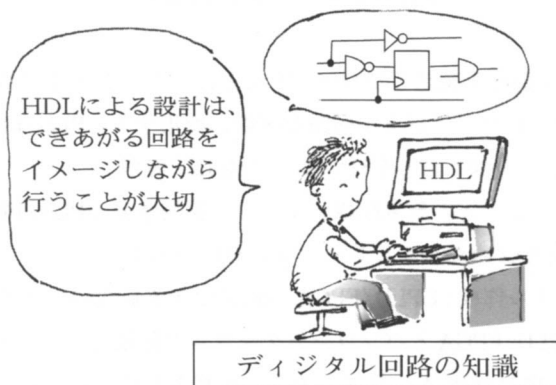



図1.28　ディジタル回路の知識は不可欠

1.4 開発環境の概要

一昔前まで，HDLでディジタル回路を設計するためには，数百万円以上の開発ツールを購入する必要がありました．しかし，現在では，ほとんどお金をかけずに開発環境を構築することができます．しかも，開発ツールは，高性能化し，使いやすさも格段に向上しています．ここでは，HDL開発で使用するツール類の概要について学びましょう．

1.4.1 開発に必要なツール

図1.29に，HDLによる回路設計で必要となるツール類を示します．

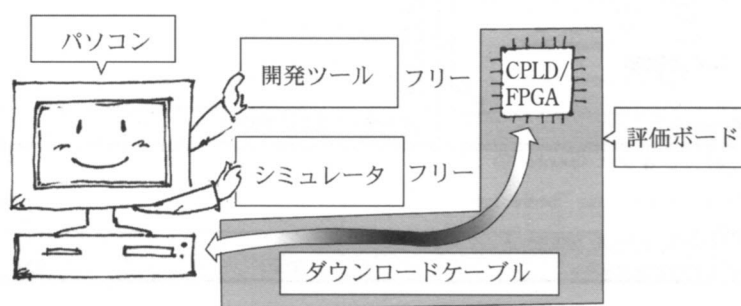



図1.29 開発に必要なツール

開発ツールやシミュレータは，WindowsやUNIX系のOSに対応したものがありますが，本書では，Windowsパソコンを使用することを前提とします．したがって，本書の手順で実習を行うためには，128 MB以上のメモリ（RAM）を搭載したWindows XPまたはWindows 2000が動作するパソコンを用意してください．また，パソコンにはパラレル（プリンタ）ポートのあることが必要です．

しかし，これらの環境が用意できなかったとしても，VHDLのソースコードは共通に使用できますから，各自の環境に合わせた手順によって実習を行うことは可能です．

つぎに，使用する開発ツール類について順に説明していきます．

1.4.2 開発ツール

HDLソースコードの記述，論理合成，配置配線，ダウンロードファイルの作成，ダウンロードなどの作業を統合的に行える開発ツールが，CPLD/FPGAメーカから提供されています．Web版は，各社のホームページからフリーでダウンロードすることができます（第7章参照）．

使用するCPLD/FPGAのメーカによって，この開発ツールを選択することになります．たとえば，アルテラ社ならQuartus IIやMAX+PLUS II，ザイリンクス社ならISE（integrated software environment）を使用します．これらの開発ツールは，VHDLだけでなく，Verilog-HDLなどにも対応しています．

本書では，主として使い勝手のよいザイリンクス社のISE WebPACKを取り上げることにしました．図1.30に，ISE WebPACKによる開発画面の例を示します．

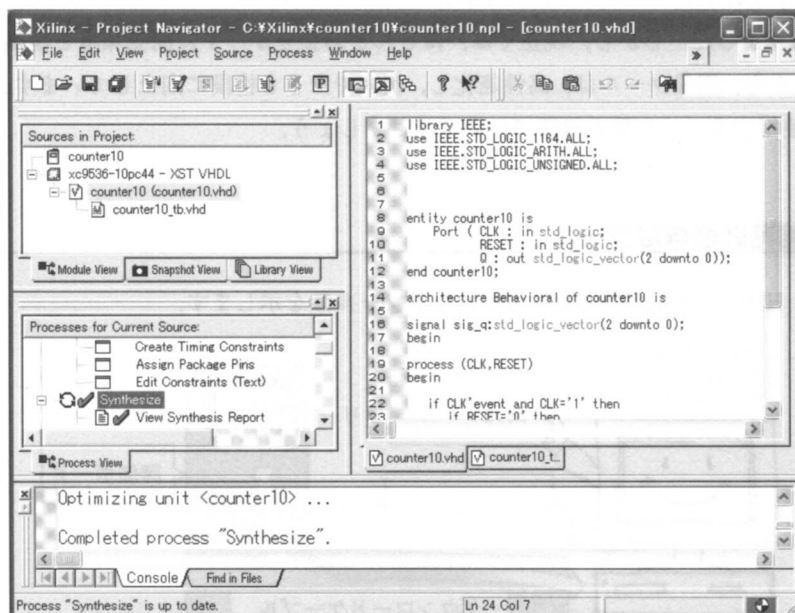



図1.30　ISE WebPACKによる開発画面の例

1.4.3　シミュレータ

シミュレータは，論理シミュレーションや遅延シミュレーションを行うソフトウェアです．シミュレータとしては，メンターグラフィックス社のModelSimが広く普及しています．アルテラ社，ザイリンクス社とも，メンターグラフィックス社と提携して，Web版のModelSimをフリーで配布しています（第6章参照）．

シミュレーションでは，設計した回路に任意の波形（クロック信号や入力信号など）を与えた場合の回路の動作を擬似的に表示します（図1.26③⑥参照）．図1.31に，ModelSimによるシミュレーション画面の例を示します．シミュレータに入力する信号を記述したファイルをテストベンチとよびます．

小規模な開発では，シミュレーションを省略して，すぐにCPLD/FPGAへのダウンロードを行い，実際に動作させることで検証を行うことも少なくありません．

本書では，ModelSim（MXEII Starter）をISE WebPACKに組み込んで統合的な操作でシミュレーションを行う方法について説明します（第6章参照）．したがって，必要なソフトウェアは

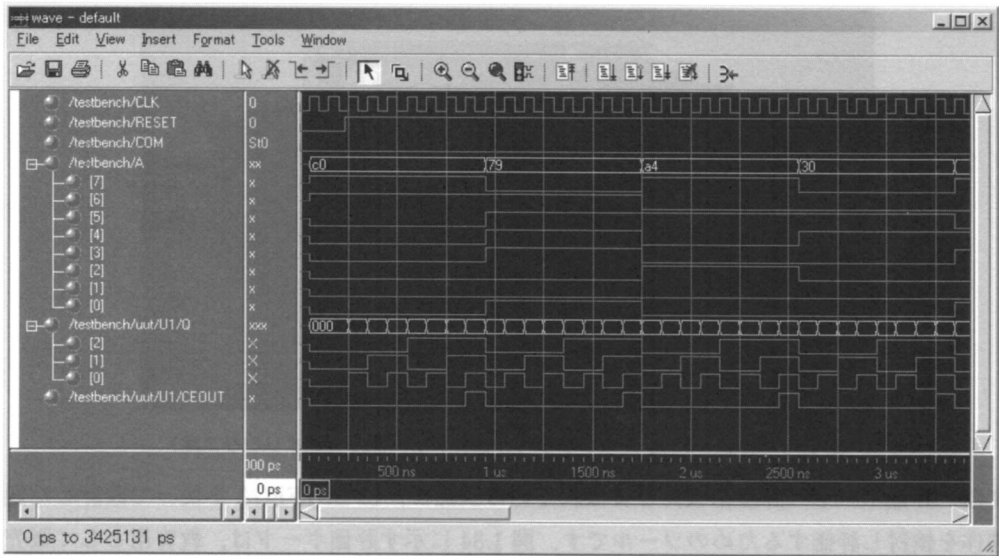



図 1.31 ModelSim によるシミュレーション画面の例

無料で用意できます.

▶1.4.4 ダウンロードケーブル

ダウンロードケーブルは, ISE WebPACK などの開発ツールによって作成したダウンロードファイルを CPLD/FPGA に転送する際に使用します. ダウンロードには, パソコンのパラレルポート（プリンタポート）を使用する JTAG（joint test action group）方式が一般的です. このためのデータ変換回路を含むダウンロードケーブルは, 各メーカから市販されています（図 1.32 参照）.

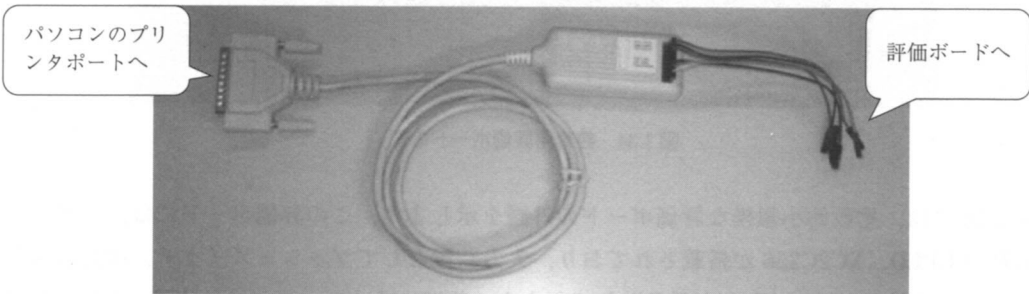



図 1.32 ダウンロードケーブルの例（Insight 製）

しかし, ダウンロードケーブル内の回路はとても簡単であり, 回路図などの情報も公開されていますので, 安く済ませたい人は自作することが可能です. また, たとえばヒューマンデータ社（http://www.hdl.co.jp）などからは, 組み立てキットや完成品が販売されています（図 1.33 参照）.

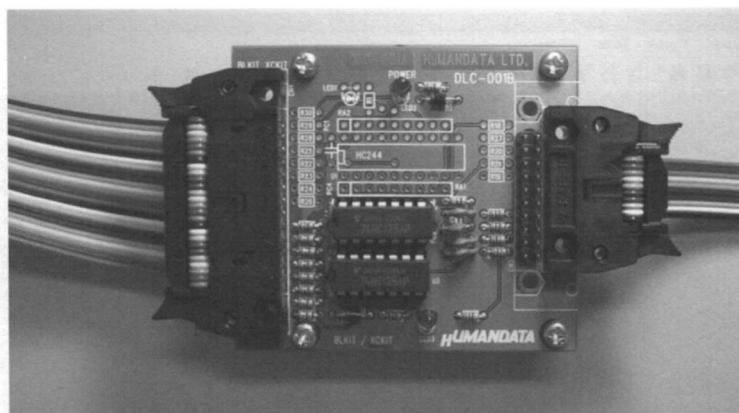



図1.33　ヒューマンデータ社のダウンロードケーブル（ザイリンクス用）

また，評価ボード（評価基板）とよばれている製品は，CPLD/FPGAを使って設計した回路の動作を検討し評価するためのツールです．図1.34に示す評価ボードは，教育用として販売されている製品の例です．

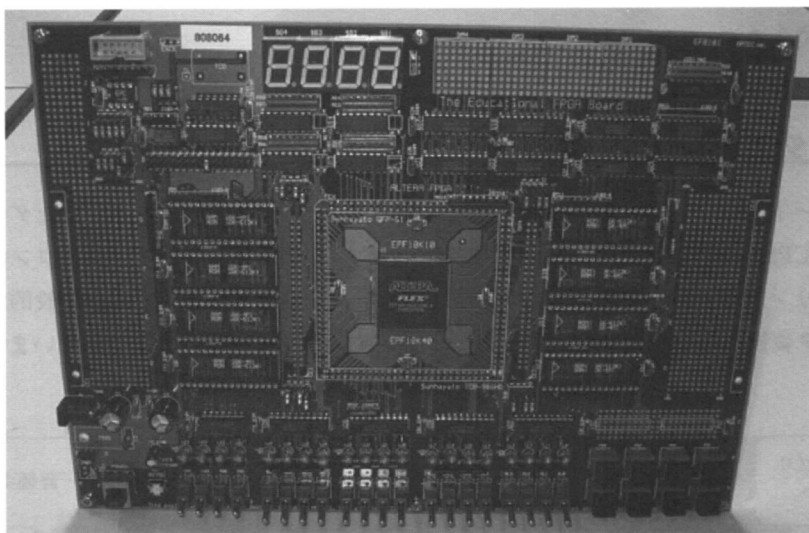



図1.34　教育用評価ボードの例

図1.35には，比較的小規模な評価ボードの外観を示します．この評価ボードには，ザイリンクス社のCPLD，XC2C256が搭載されており，入力装置としてプッシュスイッチ，出力装置として液晶表示ディスプレイなどが用意されています．基板には，ユーザが外付け回路を製作して実験を行うスペースもあります．この評価ボードは，乾電池で動作させることも可能です．

評価ボードの中には，ダウンロードに必要なJTAGインタフェース回路を搭載したものがありますので，そのような評価ボードを使用すれば図1.32や図1.33に示したダウンロードケーブルを別途用意する必要はありません．JTAGインタフェース回路を含まない，単なるケーブルでパソコンと接続すればよいのです．

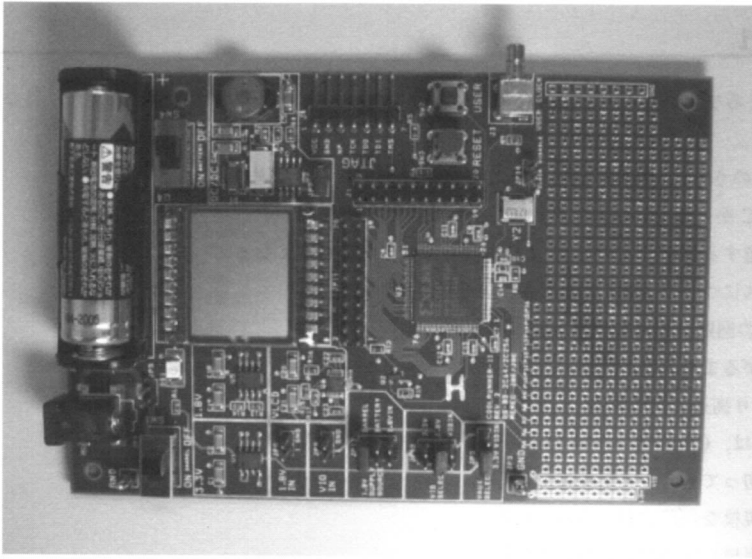



図 1.35 小規模な評価ボードの例（Memec Japan 社）

本書では，図 1.36 に示すコストパフォーマンスのすぐれた評価ボードを基本にして実習方法を説明します．この評価ボードは，ザイリンクス社の CPLD「CoolRunnerII シリーズの XC2C256」を採用しており，豊富な入出力装置の他，JTAG インタフェース回路も搭載していますから，直接パソコンと接続して各種の実習を行うことができます．入手方法は，200 ページで紹介します．

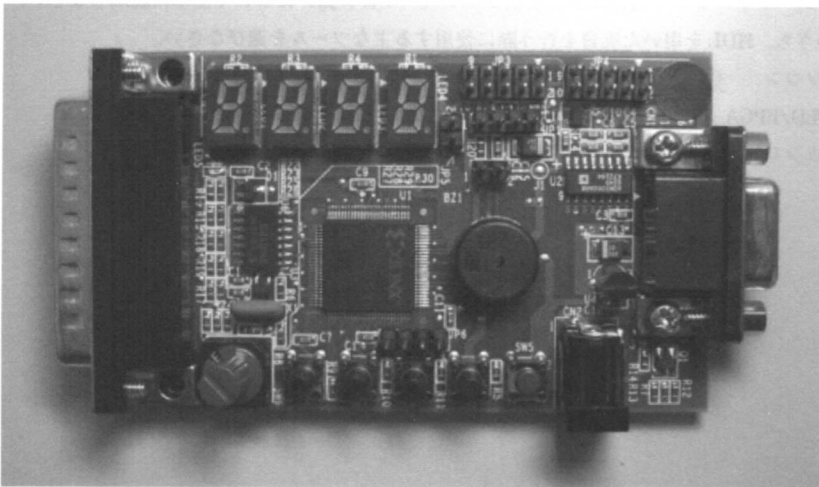



図 1.36 実習用評価ボード（ソリトンウェーブ社）

▶ 演習問題 1

1.1　CPUに与えるプログラムとCPLD/FPGAに与えるコードは，動作の仕方がどのように異なるのか簡単に説明しなさい.

1.2　つぎの文章のうち，CPLD/FPGAのもつ長所に該当するものを選びなさい.

① 1個のICから個人レベルでも仕立て上げることができる

② 大量生産する場合には，ASICより1個あたりの価格が安くなる

③ スペースについてASICより最適化が可能である

④ IC内部の回路修正や変更が容易にできる

⑤ 出来あがるまでの時間が短い

⑥ ASICより高速に動作する

1.3　つぎの文章は，CPLDとFPGAのどちらに該当するものか答えなさい.

① 電源を切ってもデータを保持できる

② 小・中規模な回路設計に向いている

③ 中・大規模な回路設計に向いている

④ ファンクションブロックという機能単位がある

⑤ CLBという機能単位がある

1.4　最も早くIEEEによって標準化されたHDLは何か.

1.5　論理シミュレーションと遅延シミュレーションの違いを簡単に説明しなさい.

1.6　テストベンチとは何か簡単に説明しなさい.

1.7　論理合成とはどのような作業か簡単に説明しなさい.

1.8　配置配線とはどのような作業か簡単に説明しなさい.

1.9　ダウンロードとはどのような作業か簡単に説明しなさい.

1.10　ダウンロードケーブルの通信規格と接続ポート（パソコン側）について簡単に説明しなさい.

1.11　つぎのうち，HDLを用いた実習を行う際に使用する主なツールを選びなさい.

① パソコン　② ワープロソフト　③ CADソフト　④ シミュレータ

⑤ CPLD/FPGAメーカの提供する開発ツール　⑥ イメージスキャナ

⑦ ダウンロードケーブル　⑧ CPLD/FPGA実装ボード　⑨ VHDLコンパイラ

第2章 設計の流れ

VHDLを用いてディジタル回路を設計する場合には，パソコンを使用した作業が不可欠になります．パソコンにVHDLのコードを入力し，論理合成や配置配線を行った後，データを実機（評価ボード）上のCPLD/FPGAにダウンロードします．

自分では理解できていると思っている事項であっても，実際に動作確認を行ってみると，予想もしなかった思い違いやトラブルが発生することは少なくありません．また，実習によって新しい発見をすることも多くあります．したがって，VHDLをマスターするためには，評価ボードを使用した実習が欠かせません．

この章では，コードの入力から評価ボードで動作確認を行うまでの流れを説明します．実際にパソコンを操作しながら，VHDLによるディジタル回路設計の流れを体験しましょう．VHDLのコードに関する詳細は，次章以降で学びます．

第2章

2.1 ターゲットデバイス

実際に使用するCPLD/FPGAをターゲットデバイスといいます．CPLD/FPGAは，各種の製品が販売されていますので，設計する回路に適合するターゲットデバイスを選択する必要があります．ここでは，ターゲットデバイス選定の考え方などについて説明します．

2.1.1 ターゲットデバイスの選定

ターゲットデバイスを選定するためには，図2.1に示すような項目について考える必要があります．

図2.1　ターゲットデバイス選定の条件

表2.1に，ザイリンクス社のCPLDの仕様を示します．ザイリンクス社のCPLDには，CoolRunner-II，CoolRunner XPLA3，XC9500XV，XC9500XLの4ファミリがあります．表2.1では，各ファミリ内の型番とその仕様が示されています．たとえば，CoolRunner-IIファミリの動作電圧は1.8 Vであり，ファミリ内のXC2C256という型番のCPLDは，6000個のシステムゲート（system gates），256個のマクロセル（macrocells）から構成されていることがわかります．ここで，システムゲート数とはCPLDを構成するゲートの総数であり，マクロセル数とはCPLD内に含まれるフリップフロップの総数を示します．ターゲットデバイスの規模を選定する場合，CPLDではマクロセル数を目安にするとよいでしょう．ただし，実際には，仕様で示されているマクロセルすべてを使用することはできませんので，何割かの余裕をもった選定を行いましょう．

XC2C256は，各種のパッケージが製品化されていますが，入出力ピン数の最大（Max.I/O）は184本です．また，表2.1で示されているスピード（speed）は，グレード（grade）で表示されており，小さい数になるほど，高速動作が可能です（15ページ図1.21参照）．たとえば，XC2C256のグレード「5」では最大周波数238 MHzでの動作が可能です．

表 2.1 CPLD の仕様（ザイリンクス社）

						I/O Features		Speed			Clocking	
	System Gates	Macrocells	Product tems per Macrocell	Input Voltage Compatible	Output Voltage Compatible	Max. I/O	I/O Banking	Min. Pin-to-Pin Logic Delay(ns)	Commercial Speed Grades (fastest to slowest)	Industrial Speed Grades (fastest to slowest)	Global Clocks	Product Term Clocks per Function Block
CoolRunner-II Family — 1.8 Volt										I/O 1.8[V],2.5[V],3.3[V]		
XC2C32	750	32	40	1.5/1.8/2.5/3.3	1.5/1.8/2.5/3.3	33	1	3.5	-3 -4 -6	-6	3	17
XC2C64	1500	64	40	1.5/1.8/2.5/3.3	1.5/1.8/2.5/3.3	64	1	4	-4 -5 -7	-7	3	17
XC2C128	3000	128	40	1.5/1.8/2.5/3.3	1.5/1.8/2.5/3.3	100	2	4.5	-4 -6 -7	-7	3	17
XC2C256	6000	256	40	1.5/1.8/2.5/3.3	1.5/1.8/2.5/3.3	184	2	5	-5 -6 -7	-7	3	17
XC2C384	9000	384	40	1.5/1.8/2.5/3.3	1.5/1.8/2.5/3.3	240	4	6	-6 -7 -10	-10	3	17
XC2C512	12000	512	40	1.5/1.8/2.5/3.3	1.5/1.8/2.5/3.3	270	4	6	-6 -7 -10	-10	3	17
CoolRunner XPLA3 Family — 3.3 Volt										I/O 3.3[V],5[V]		
XCR3032XL	750	32	48	3.3/5	3.3	36		5	-5 -7 -10	-7 -10	4	16
XCR3064XL	1500	64	48	3.3/5	3.3	68		6	-6 -7 -10	-7 -10	4	16
XCR3128XL	3000	128	48	3.3/5	3.3	108		6	-6 -7 -10	-7 -10	4	16
XCR3256XL	6000	256	48	3.3/5	3.3	164		7.5	-7 -10 -12	-10 -12	4	16
XCR3384XL	9000	384	48	3.3/5	3.3	220		7.5	-7 -10 -12	-10 -12	4	16
XCR3512XL	12000	512	48	3.3/5	3.3	260		7.5	-7 -10 -12	-10 -12	4	16
XC9500XV Family — 2.5 Volt										I/O 2.5[V],3.3[V]		
XC9536XV	800	36	90	2.5/3.3	1.8/2.5/3.3	36	1	5	-5 -7	-7	3	18
XC9572XV	1600	72	90	2.5/3.3	1.8/2.5/3.3	72	1	5	-5 -7	-7	3	18
XC95144XV	3200	144	90	2.5/3.3	1.8/2.5/3.3	117	2	5	-5 -7	-7	3	18
XC95288XV	6400	288	90	2.5/3.3	1.8/2.5/3.3	192	4	6	-6 -7 -10	-7 -10	3	18
XC9500XL Family — 3.3 Volt										I/O 3.3[V],5[V]		
XC9536XL	800	36	90	2.5/3.3/5	2.5/3.3	36		5	-5 -7 -10	-7 -10	3	18
XC9572XL	1600	72	90	2.5/3.3/5	2.5/3.3	72		5	-5 -7 -10	-7 -10	3	18
XC95144XL	3200	144	90	2.5/3.3/5	2.5/3.3	117		5	-5 -7 -10	-7 -10	3	18
XC95288XL	6400	288	90	2.5/3.3/5	2.5/3.3	192		6	-6 -7 -10	-7 -10	3	18

XC95--XL =9500 3.3V
XC95--XV =9500 2.5V
XCR-XL =CoolRunner 3.3V
XC2C =CoolRunner II 1.8V

表 2.2 に，CPLD のパッケージ一覧（ザイリンクス社）を示します．同じ機種であっても PQ (plastic quad flat pack) や TQ (thin quad flat pack) など各種のパッケージ形状があります．また，同じパッケージ形状でも，入出力ピン数の違いでピンの総数が異なるデバイスがあります．たとえば，XC2C256 では，PQ，VQ，TQ，CP，FT のパッケージ形状が用意されており，それらの「入出力ピン数/ピン総数」は，それぞれ，173/208，80/100，118/144，106/132，184/256 本となっています．

表2.2　CPLDのパッケージ一覧（ザイリンクス社）

		CoolRunner-II						CoolRunner XPLA3						XC9500XV				XC9500XL			
	Body Size	XC2C32	XC2C64	XC2C128	XC2C256	XC2C384	XC2C512	XCR3032XL	XCR3064XL	XCR3128XL	XCR3256XL	XCR3384XL	XCR3512XL	XC9536XV	XC9572XV	XC95144XV	XC95288XV	XC9536XL	XC9572XL	XC95144XL	XC95288XL
PLCC Packages (PC)																					
44	16.5 × 16.5 mm	33	33					36	36					34	34			34	34		
PQFP Packages (PQ)																					
208	28 × 28 mm				173	173	173				164	172	180				168				168
VQFP Packages (VQ)																					
44	12 × 12 mm	33	33					36	36					34	34			34	34		
64	12 × 12 mm																	36	52		
100	16 × 16 mm		64	80	80				68	84											
TQFP Packages (TQ)																					
100	14 × 14 mm														72	81			72	81	
144	20 × 20 mm			100	118	118				108	120	118				117	117			117	117
Chip Scale Packeges (CP) — wire-bond chip-scale BGA (0.5 mm ball spacing)																					
56	6 × 6 mm	33	45						48												
132	8 × 8 mm			100	106																
Chip Scale Packeges (CS) — wire-bond chip-scale BGA (0.8 mm ball spacing)																					
48	7 × 7 mm							36	40					36	38			36	38		
144	12 × 12 mm									108						117				117	
280	16 × 16 mm										164						192				192
BGA Packeges (BG) — wire-bond standard BGA (1.27 mm ball spacing)																					
256	27 × 27 mm																				192
FGA Packeges (FT) — wire-bond fine-pitch thin BGA (1.0 mm ball spacing)																					
256	17 × 17 mm				184	212	212				164	212	212								
FBGA Packeges (FG) — wire-bond Fineline BGA (1.0 mm ball spacing)																					
256	17 × 17 mm																192				192
324	23 × 23 mm					240	270					220	260								

Automotive products are highlighted :（自動車用）
-40C to + 125C ambient temperature for CPLDs

Package Type :

BG	= Ball Grid Array	CS	= Chip Scale
FG	= Fine-Pitch Ball Grid Array	CG	= Chip Grid Array
PQ	= Plastic Quad Flat Pack	PC	= PLCC
HQ	= High Heat Dissipation QFP	CP	= Chip Scale
TQ	= Thin Quad Flat Pack	FF	= Flip Chip
VQ	= Very Thin Quad Flat Pack	FT	= Plastic Fine-Pitch BGA

異なる機種であっても，同じパッケージ形状で同じ総ピン数をもったCPLDでは，ピン配置に互換性があります．したがって，たとえば256個のマクロセルを搭載したXC2C256を使うつもりで基板を製作した後に，何かの理由でそれ以上のマクロセルを搭載したCPLDに変更する必要が生じた場合でも，基板構成を変更することなくCPLDを差し替えることが可能です．たとえば，XC2C256のTQ144とXC2C384のTQ144などが差し替え可能です．

表2.3にザイリンクス社FPGA，Spartan-3ファミリの仕様，表2.4にパッケージ一覧を示します．FPGAでは，規模の選定にシステムゲート，CLB，スライスなど（15ページ図1.22参照）

表 2.3　FPGA の仕様（ザイリンクス社 Spartan-3 ファミリ）

		CLB Resaurces			Memory Resaurces			DSP	Clock Resaurces		I/O Features				Speed		
	System gates (see note 1)	CLB Array (Row X Col)	Number of Slices	Logic Cells (see note 2)	Max. Distributed RAM (Bits)	# 18 kbits Block RAM	Total Block RAM (Bits)	# 18 × 18 Dedicated Multipliers	DCM Frequency (min/max)	# DCM Blocks	Digitaly Controlled Impedance	Maximum Differential I/O Pairs	Max. I/O	I/O Standards	Commercial Speed Grades (slowest to fastest)	Industrial Speed Grades (slowest to fastest)	Config. Memory (Bits)
Spartan-3	Family1.2V					I/O	1.2[V],1.5[V],1.8[V],2.5[V],3.3[V]						90nm Eight Layer Metal Process				
XC3S50	50K	16 × 12	768	1,728	12K	4	72K	4	25/325	2	YES	56	124	Single-ended	-4 -5	-4	0.3M
XC3S200	200K	24 × 20	1,920	4,320	30K	12	216K	12	25/325	4	YES	76	173	LVTTL,LVCMOS3.3/2.5/1.8/1.5/1.2	-4 -5	-4	1.0M
XC3S400	400K	32 × 28	3,584	8,064	56K	16	288K	16	25/325	4	YES	116	264	PCI3.3V-32/64-bit 33MHz,	-4 -5	-4	1.7M
XC3S1000	1000K	48 × 40	7,680	17,280	120K	24	432K	24	25/325	4	YES	175	391	SSTL2 Class I & II,SSTL18Class I,	-4 -5	-4	3.2M
XC3S1500	1500K	64 × 52	13,312	29,952	208K	32	576K	32	25/325	4	YES	221	487	HSTL Class I,II,HSTL1.8 Class I,II & III,GTL,GTL+	-4 -5	-4	5.2M
XC3S2000	2000K	80 × 64	20,480	46,080	320K	40	720K	40	25/325	4	YES	270	565	Differential	-4 -5	-4	7.7M
XC3S4000	4000K	96 × 72	27,648	62,208	432K	96	1,728K	96	25/325	4	YES	312	712	LVDS2.5,BusLVDS2.5,UltraLVDS2.5,LVDS_ext2.5	-4 -5	-4	11.3M
XC3S5000	5000K	104 × 80	33,280	74,880	520K	104	1,872K	104	25/325	4	YES	344	784	RSDS,LDT2.5	-4 -5	-4	13.3M

注 : speed grade は，数値が大きい程高速になります（CPLD とは逆）.

表 2.4　FPGA のパッケージ一覧（ザイリンクス社 Spartan-3 ファミリ）

			Spartan-3 (1.2V)							
			XC3S50	XC3S200	XC3S400	XC3S1000	XC3S1500	XC3S2000	XC3S4000	XC3S5000
Pins	Body Size	I/O's	124	173	264	391	487	565	712	784
PQFP Packages (PQ)										
PQ208	30.6 × 30.6 mm		124	141	141					
VQFP Packages (VQ)										
100	16 × 16 mm		63	63						
TQFP Packages (TQ)										
TQ144	22 × 22 mm		97	97	97					
FGA Packeges (FT) — wire-bond fine-pitch thin BGA (1.0 mm ball spacing)										
FT256	17 × 17 mm			173	173	173				
FGA Packeges (FG) — wire-bond fine-pitch BGA (1.0 mm ball spacing)										
FG456	23 × 23 mm				264	333	333			
FG676	27 × 27 mm					391	487	489		
FG900	31 × 31 mm							565	633	633
FG1156	35 × 35 mm								712	784

の数を指標にするとよいでしょう.

FPGA は，集積度が高く大規模な回路に用いられます．表 2.1 と表 2.3 から，CPLD と FPGA のシステムゲート数を比較すると FPGA の規模の大きさがわかることでしょう.

▶2.1.2　CPLD の例

ザイリンクス社 CPLD の最新ファミリは，CoolRunner-II です．この CPLD の主な特徴はつぎのとおりです.

① 低電圧動作　→　動作電圧 1.8 V

② 低消費電力　→　消費電力 10 mW（XC2C128 で 8 個の 16 ビットカウンタを 50 MHz で動作させた場合），スタンバイ時 180 μW

③ 複数のI/Oスタンダード　→　入出力ピンでは，1.5 V/1.8 V/2.5 V/3.3 Vの各電圧を扱うことが可能

④ 高速動作　→　最大動作周波数385 MHz

⑤ 最大マクロセル512個　→　ファンクションブロック1個あたり，マクロセル16個

⑥ 多いI/Oピン数　→　マクロセル数あたりの入出力ピン数が多い

⑦ 4レベルのセキュリティ　→　パターンの盗用を防ぐ

ここでは，CoolRunner-IIファミリに属する「XC2C256-7VQ100C」についてみてみましょう．図2.2に，CPLDの外観とピン配置を示します．型番より，このCPLDは256個のマクロセルをもつことや，薄型のパッケージVQ（very thin quad flat pack）形状をしており，ピン総数は100本であることなどがわかります．

（a）外観

（b）ピン配置

図2.2　XC2C256-7VQ100C

さらに，表2.2から「XC2C256-7VQ100C」を調べると，ボディサイズ（body size）は16×16 mm，入出力ピン数は80本，自動車用途など周囲温度－40～＋125℃での動作が可能なことがわかります．また，スピードグレードは「7」であり，200 MHzを超える周波数で動作させることも可能です（データシートより）．消費電力は，動作周波数によって大きく異なりますので注意してください．たとえば，220 MHzで動作させる場合には，30 MHzの場合と比べて約7倍の消費電力を必要とします．

価格は，購入店や購入個数によって異なるでしょうが，筆者が正規代理店から1個を購入した場合は3100円でした．

2.1.3 評価ボードの例

たとえば，10 進数で 0 ～ 65535 までのカウントを行う 16 ビットカウンタを構成する場合には，16 個のマクロセル（フリップフロップ）を使用します．このように，簡単なディジタル回路を構成して VHDL の実習を行う場合，256 個のマクロセルを備えた「XC2C256-7VQ100C」は十分な規模といえるでしょう．また，CPLD は，フラッシュメモリを搭載していますので，電源を切っても回路構成を保持できる利点があります．

以上のような観点から，本書では「XC2C256-7VQ100C」を搭載したソリトンウェーブ社の「HDL トレーナー（HDL-10）」という評価ボードを使用して実習を行うことにします．図 2.3 に，HDL トレーナーの外観を示しますが，詳細は第 7 章 195 ページを参照してください．

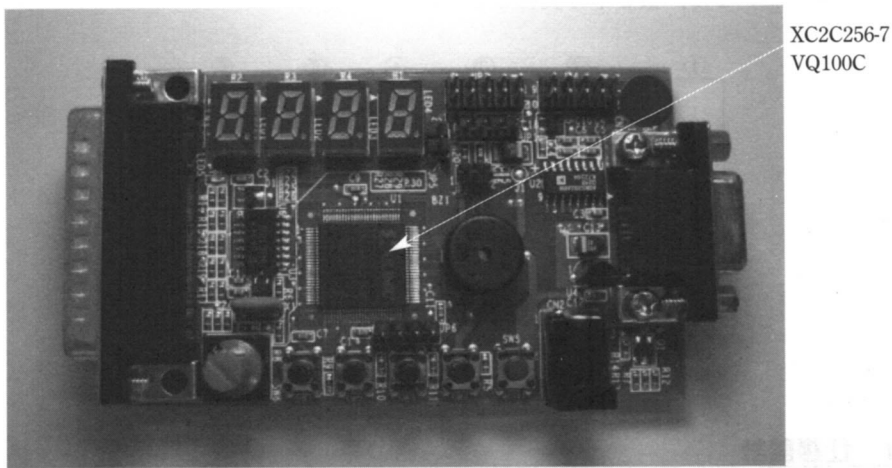



図 2.3 HDL トレーナーの外観

異なる評価ボードを使用する場合には，CPLD/FPGA の各ピンに接続されている入出力装置（スイッチや LED）などの仕様を説明書により確認してください．

2.2 実装までの手順

ここでは，開発ツールを実際に操作して，VHDLで記述したコードから論理合成，配置配線を行い，CPLDへダウンロードして動作させるまでの過程を実習します．図2.4で実習の流れを確認しましょう（19ページ図1.26参照）．実習環境としては，開発ツールにザイリンクス社のISE WebPACK，評価ボードにソリトンウェーブ社のHDLトレーナーを使用します．なお，シミュレーションについては，第6章で実習します．

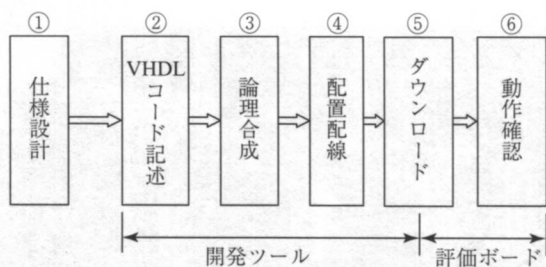



図2.4　実習の流れ

2.2.1 仕様設計

ここでの実習では，評価ボードに搭載されている7セグメントLEDを0～9まで順次表示することを繰り返す回路を設計しましょう．ただし，リセットスイッチ（プッシュスイッチ）を押すとリセットがかかり，表示は0から再スタートすることとします．図2.5に，この実習回路の構成を示します．実習回路は，分周回路，10進カウンタ回路，デコーダ回路の三つに大別できます．

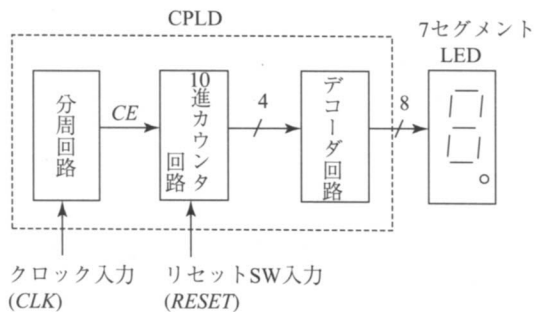



図2.5　実習回路の構成

10 進カウンタ回路では，2 進数の 0000 ～ 1001（10 進数の 0 ～ 9）を繰り返しカウントします．カウンタの出力は，デコーダ回路によって 7 セグメント LED で数字を表示するデータにデコードされます．一方，使用する評価ボードは，4 MHz のセラロックを動作クロック源としています．したがって，そのまま動作させせると，式（2.1）のように 0.25 μs の周期 T で 7 セグメント LED が変化するため速すぎて目では確認できません．

$$T=\frac{1}{f}=\frac{1}{4\times10^{6}}=0.25\times10^{-6}\text{s}=0.25\ \mu\text{s} \qquad (2.1)$$

そこで，10 進カウンタ回路の動作を遅くするために分周回路を設けます．22 ビットの分周回路によって 4 MHz を分周すると，その周期 T' は式（2.2）のようにおよそ 1 秒になります．

$$T'=(0.25\times10^{-6})\times2^{22}=1.049\ \text{s} \qquad (2.2)$$

つまり，4 MHz のクロック *CLK* が 2^{22} 回発生するたびに，新しいクロック *CE* を 1 個作ります．そして，10 進カウンタ回路は，クロック *CE* によって動作させるのです．

また，リセット信号 *RESET* は，クロック *CLK* に無関係に作用する非同期リセットとします．

▶2.2.2　VHDL コードの記述

回路の仕様設計が終わったので，開発ツールを使って VHDL のコードを記述します．ここでは，例としてザイリンクス社の ISE WebPACK6.1i を用いた手順を説明します．バージョンによっては，手順や画面に違いがあることと思います．なお，ISE WebPACK の入手やインストールの方法については，第 7 章を参照してください．

① ISE WebPACK の起動

図 2.6 に示す ISE WebPACK のアイコンをダブルクリックし，図 2.7 に示す起動画面を表示します．

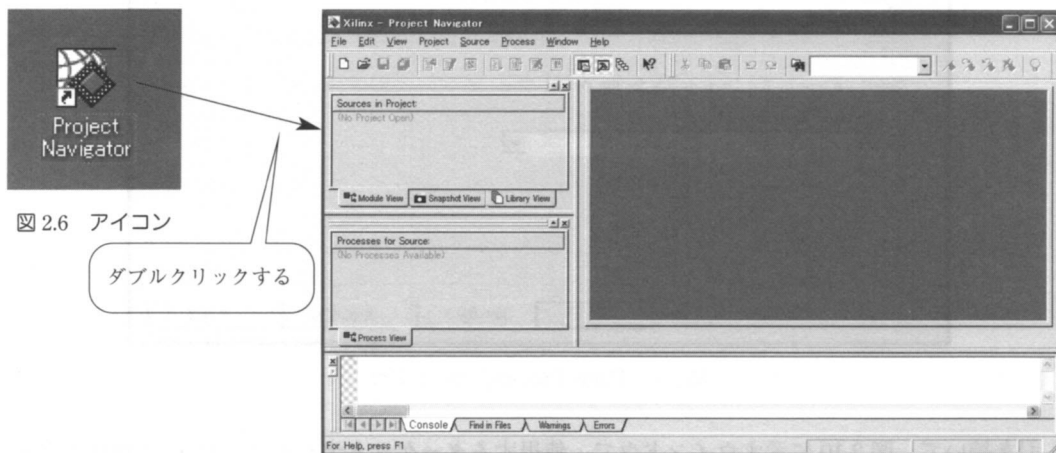



図 2.6　アイコン

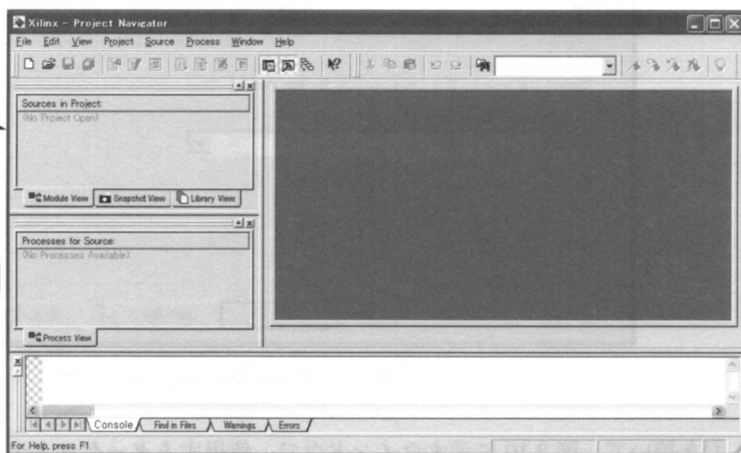



図 2.7　起動画面

② プロジェクトの準備

ISE WebPACK では，各種のデータをプロジェクト（Project）という単位で管理します．した

がって，始めにプロジェクトの作成準備を行います．今回の実習のプロジェクト名は，「LED10」とします．図2.8に示すように，ツールバーの「File」→「New Project」を選択します．

図2.8　「New Project」を選択

図2.9に示すウインドウが現れますので，「Project Name」欄に「LED10」と入力します．「Project Location」欄にはプロジェクトを保存する場所を指定します．「Top-Level Module Type」は，「HDL」とし，「次へ」ボタンをクリックします．

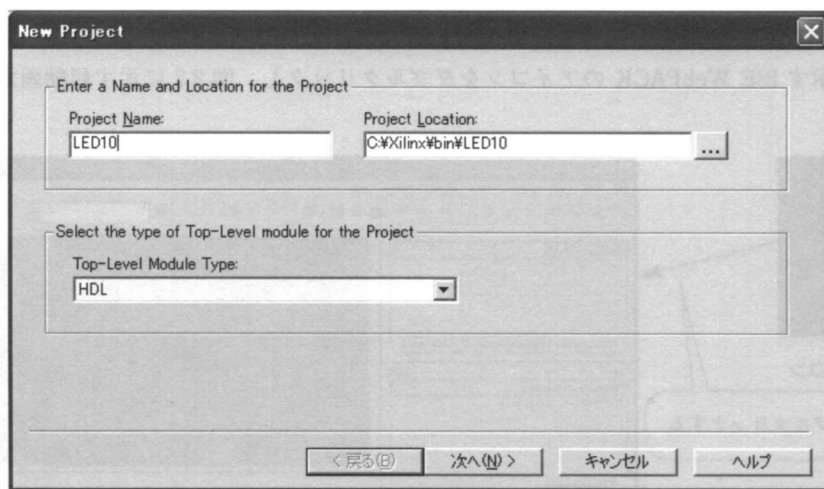



図2.9　「New Project」ウインドウ

引き続いて，図2.10に示すウインドウで，使用するターゲットデバイスやHDLの種類などを指定します．図2.10のように入力したら，「次へ」ボタンをクリックします．

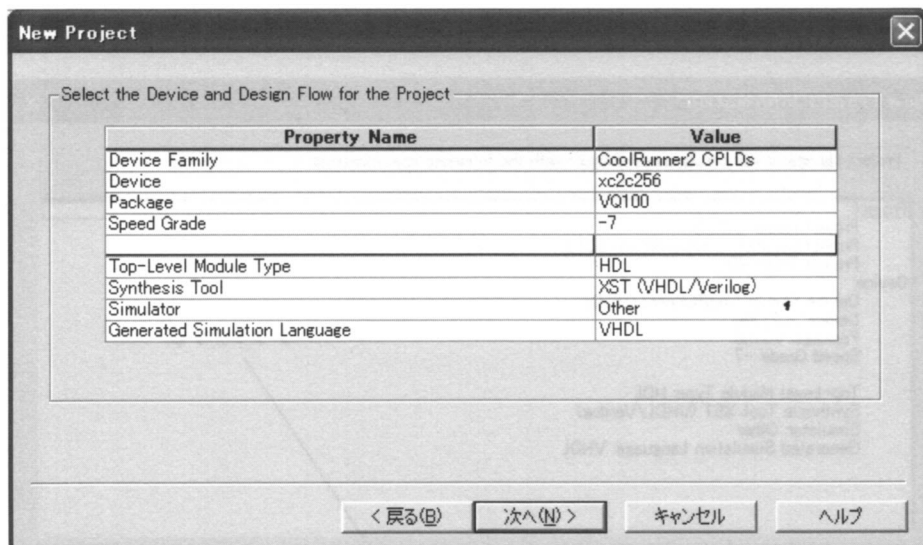



図 2.10　ターゲットデバイスや HDL の種類を指定

続いて，図 2.11，2.12 に示すウインドウが現れますが，ここではそのまま「次へ」をクリックして先に進みます．

New Project
Create a New Source
Source File | Type
New Source...
Remove
Create a new source to add to the project (optional). Only one new source can be specified now. Additional new sources can be added after project creation using the "Project->New Source" command
< 戻る(B)　次へ(N) >　キャンセル　ヘルプ

クリックする

図 2.11　「Create a New Source」

クリックする

New Project
Add Existing Sources
Source File | Type | Copy to Project
Add Source...
Remove
Add existing sources to the project (optional). Additional sources can be added after project creation using the "Project->Add Source" or "Project->Add Copy of Source" commands.
< 戻る(B)　次へ(N) >　キャンセル　ヘルプ

図 2.12　「Add Existing Sources」

図 2.13 に示すウインドウで，設定内容を確認したら「完了」ボタンをクリックします．

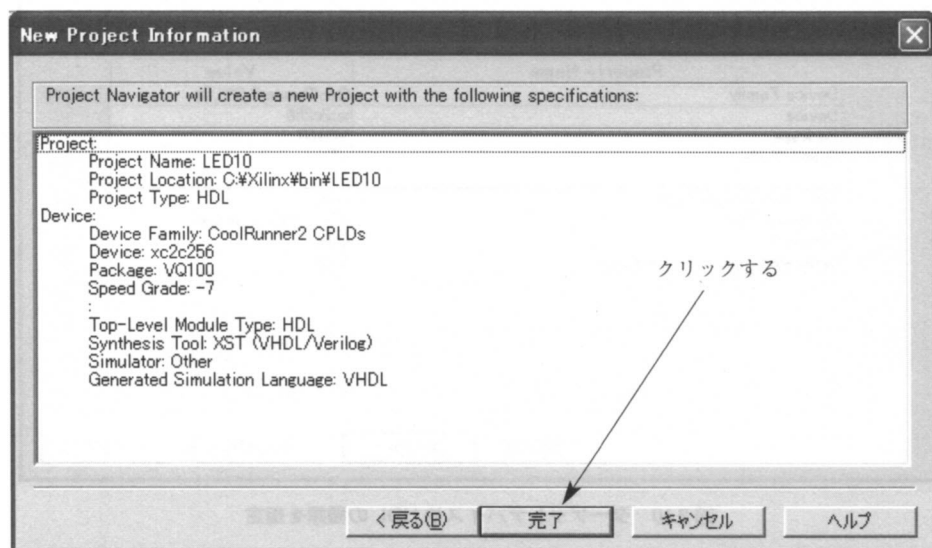



図 2.13　確認用ウインドウ

以上の操作で，プロジェクトの作成準備が終わりました．図 2.14 に示すように，「Module View」ウインドウにプロジェクト「LED10」が追加されているはずです．

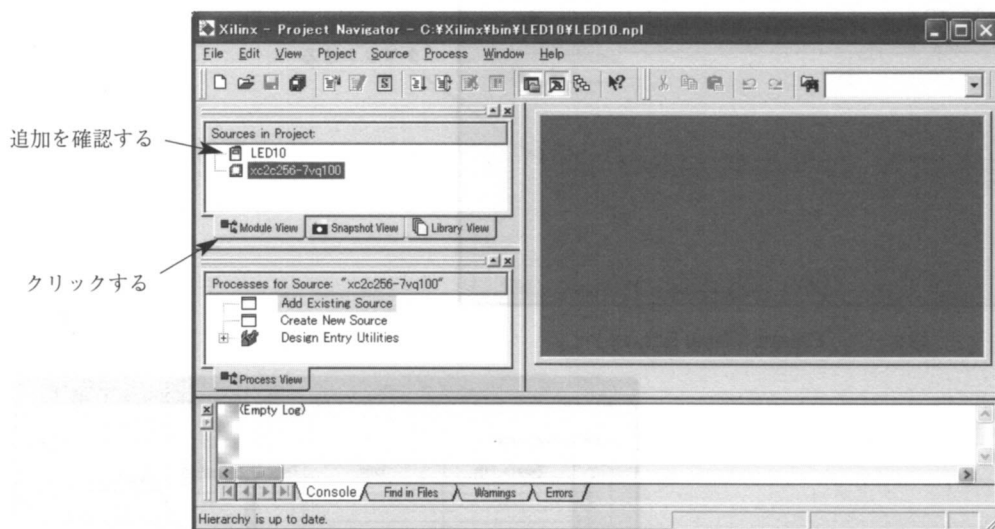



図 2.14　プロジェクトの作成準備終了

③ コード入力の準備

図 2.15 に示すように，ツールバーの「Project」→「New Source」を選択します．

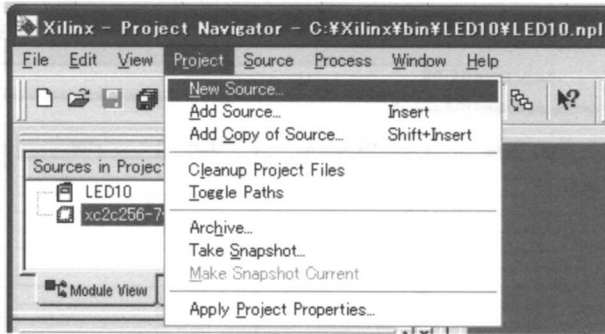



図 2.15　「New Source」を選択

図 2.16 に示すウインドウが現れますので，ウインドウ内の左欄から「VHDL Module」を選択し，「File Name」に「LED10」と入力して「次へ」ボタンをクリックします．

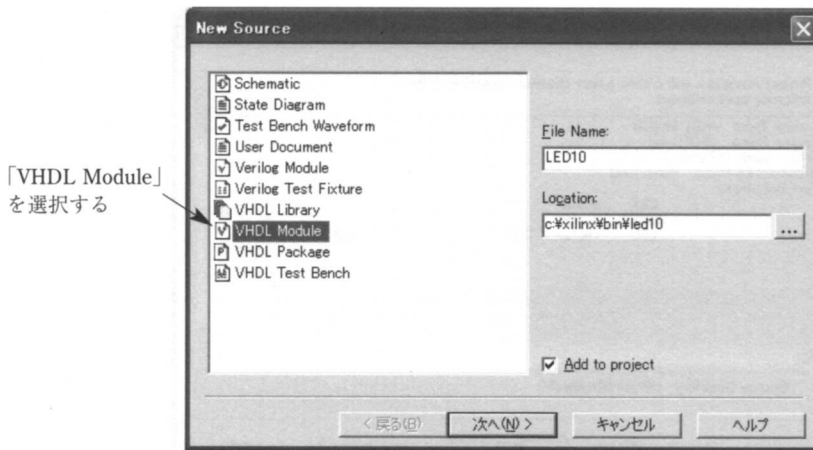



図 2.16　「New Source」ウインドウ

すると，実習回路の入出力ピンを指定するウインドウが現れますので，図 2.17 に示すような設定を行いましょう．実習回路では，入力ピンがクロック CLK，リセット RESET の各 1 本，出力ピン LED が 0 から 7 までの 8 本です．「Direction」欄では，入力ピンに「in」，出力ピンに「out」を指定します．設定後に「次へ」ボタンをクリックすると，図 2.18 に示す確認ウインドウが現れますので，設定した内容を確認して「完了」ボタンをクリックします．

以上でコード入力の準備が整いました．ISE WebPACK のウインドウには，図 2.19 に示すように，「LED10.vhd」というファイル名のコード入力用ウインドウが表示されているはずです．このように，VHDL のコードを記述するファイルは，拡張子「vhd」を付ける決まりになっています．また，表示されているコード入力用ウインドウには，すでに VHDL のコードの一部が表示されています．これは，図 2.17 で指定した入出力ピンの情報などをもとに，開発ツールが自動的に必要な VHDL コードを生成したためです．この機能を活用すれば，コード入力の負担を軽減することができます．もちろん，入出力ピンなどの割り当ては，後で自由に変更することが可能です．

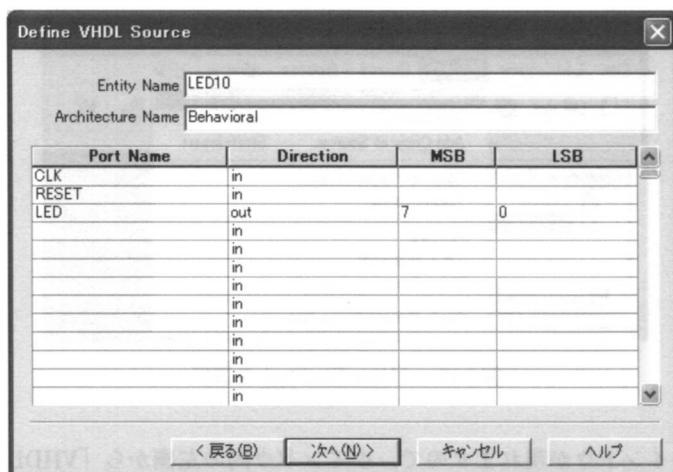



図 2.17　入出力ピンの設定

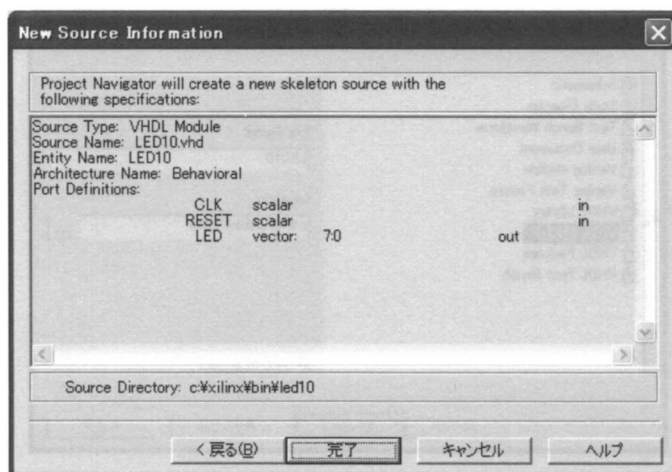



図 2.18　確認用ウインドウ

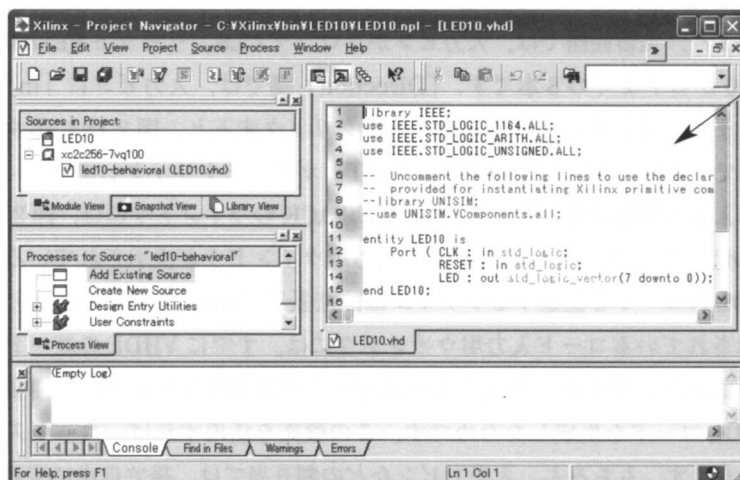



図 2.19　コード入力の準備終了

④ コードの入力

入力する実習回路の VHDL コードをリスト 2.1 に示します．ここでは，コードの詳細についての理解は必要ありません．

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
--library UNISIM;
--use UNISIM.VComponents.all;

entity LED10 is
    Port ( CLK : in std_logic;
           RESET : in std_logic;
           LED : out std_logic_vector(7 downto 0));
end LED10;

architecture Behavioral of LED10 is

 begin

 signal CNT: std_logic_vector(3 downto 0);
 signal Q : std_logic_vector(21 downto 0);
 signal CE : std_logic;
        begin

        -- 分周回路　２２ビット
        process (CLK)
        begin

           if CLK'event and CLK='1' then
                Q <= Q + '1';
           end if;
        end process;

        process (Q)
        begin
           if Q = "1111111111111111111111" then
              CE <= '1';
           else
              CE <= '0';
           end if;
        end process;

        -- 10 進カウンタ回路
        process(RESET,CE)
            begin
                if(RESET = '0') then
                       CNT <= "0000";
                   elsif (CE'event and CE='1') then
                      if(CNT = "1001") then
                                 CNT <= "0000";
```

コメント文
（以降のコードでは
省略します）

ここまでは自動的に
入力されている

ここから下を新たに
入力する

```
                    else
                          CNT <= CNT + '1';
                end if;
          end if;
      end process;

  -- デコーダ回路
  process(CNT)
      begin
            case CNT is
                      when "0000" => LED <= "10000001";
                      when "0001" => LED <= "11001111";
                      when "0010" => LED <= "10010010";
                      when "0011" => LED <= "10000110";
                      when "0100" => LED <= "11001100";
                      when "0101" => LED <= "10100100";
                      when "0110" => LED <= "10100000";
                      when "0111" => LED <= "10001101";
                      when "1000" => LED <= "10000000";
                      when "1001" => LED <= "10000100";
                      when others => LED <= "11111111";
              end case;
  end process;

  end Behavioral;
```

ここから上を新たに入力する

リスト2.1　実習回路のVHDLコード

リスト2.1の初めの部分は，コード自動生成機能によってすでに入力済みですから，残りの部分を入力します．もちろん実際にキーボードから入力することもできますが，ここでは流れを体験することが目的なので，筆者の提供するソースファイルから未入力部分をコピーするとよいでしょう（ワードパッドなどのエディタを使用して，コピーと貼り付けを行う）．リスト2.1を確認しながら，未入力部分を埋めてください．または，コード入力用ウインドウの表示内容をすべてクリアしてから，入手したソースファイルの内容すべてを貼り付けても結構です．ソースファイルの入手方法は，まえがきのページを参照してください．

図2.20　コード入力用ウインドウ

上書き保存ボタン　コピーボタン　貼り付けボタン

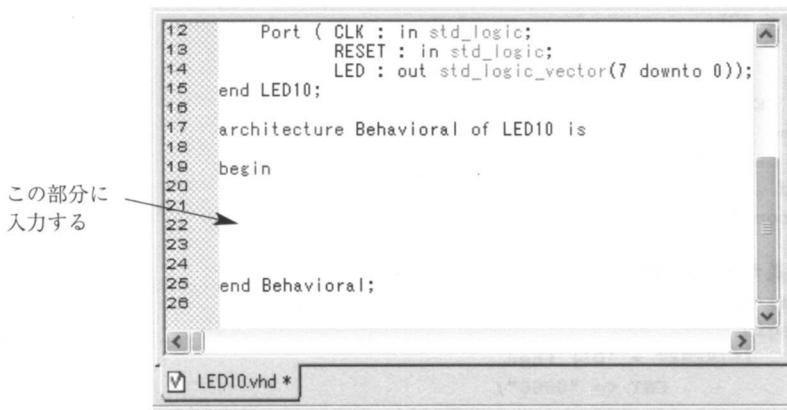
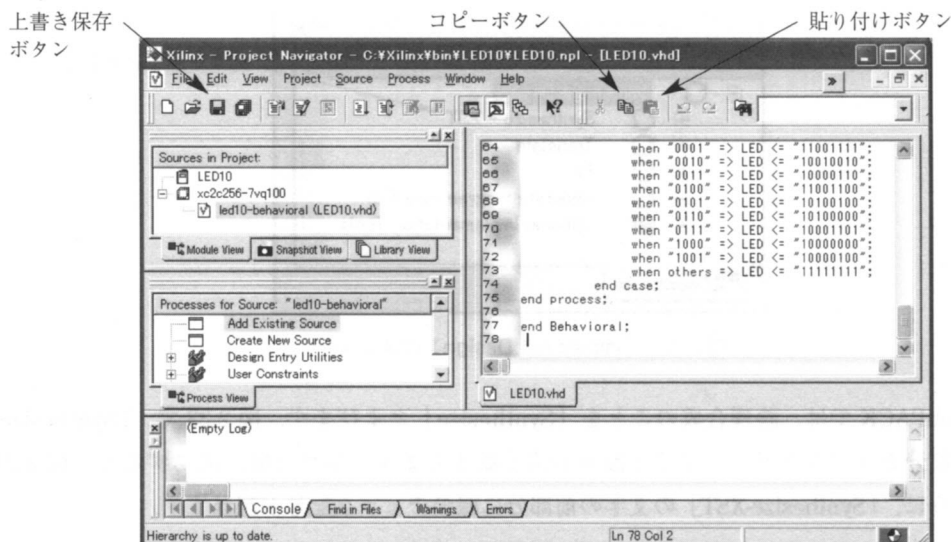



図 2.21　コードの入力を終えたウインドウ

コードの入力が終わったら，上書き保存ボタンをクリックしてコードを保存しておきましょう（図 2.21 参照）．

▶2.2.3　論理合成

論理合成などを行うメニューは，「Process View」ウインドウの「Implement Design」の中に収められています．図 2.22 に示すように，「Implement Design」の「+」記号をクリックして，「−」記号に変化させるとメニューが表示されます（図 2.23 参照）．

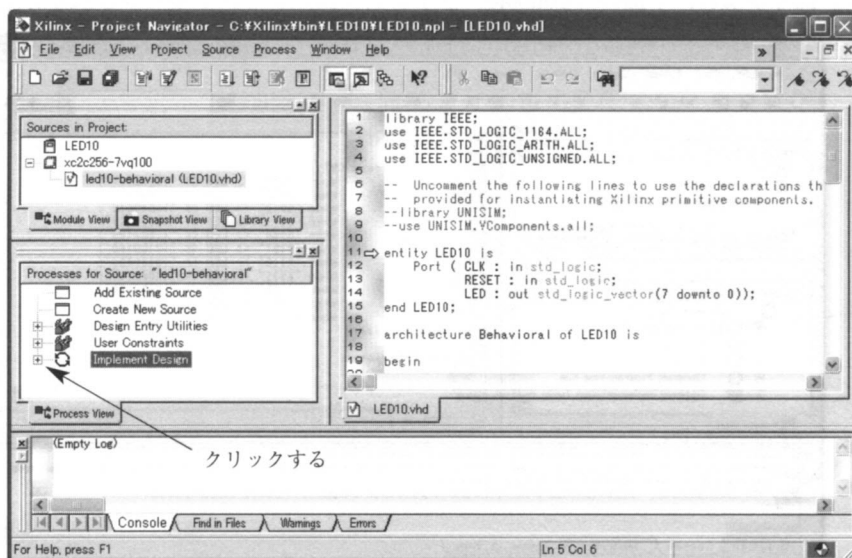



図 2.22　「Implement Design」の「+」記号をクリック

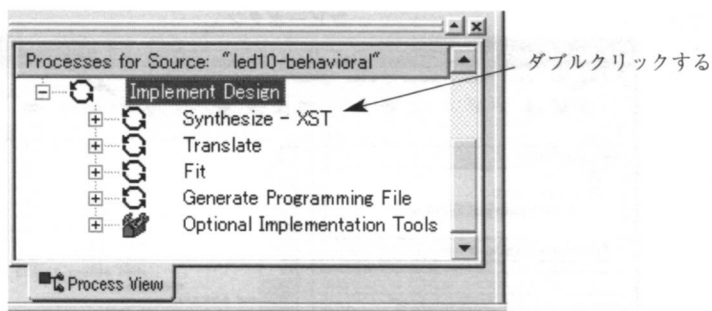



図 2.23　「Implement Design」のメニュー

ISE WebPACK では，論理合成のことを「Synthesize」とよびます．図 2.23 で，「Synthesize-XST」の部分をダブルクリックすると論理合成が始まります．論理合成が成功すると，図 2.24 に示すように，「Synthesize-XST」の文字の前部分に緑のチェック記号が現れます．

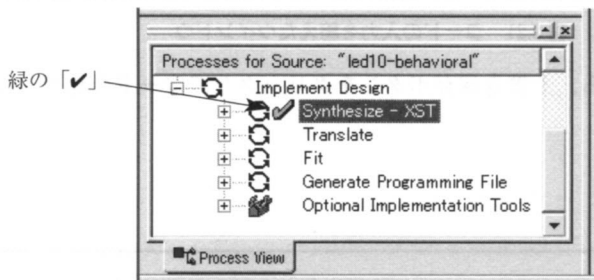



図 2.24　論理合成の成功

もし，コードにエラー（ERROR）があり，論理合成に失敗した場合には，図 2.25 に示すように赤い「×」記号が現れ，下部の「Console」ウインドウにエラーメッセージが表示されます．

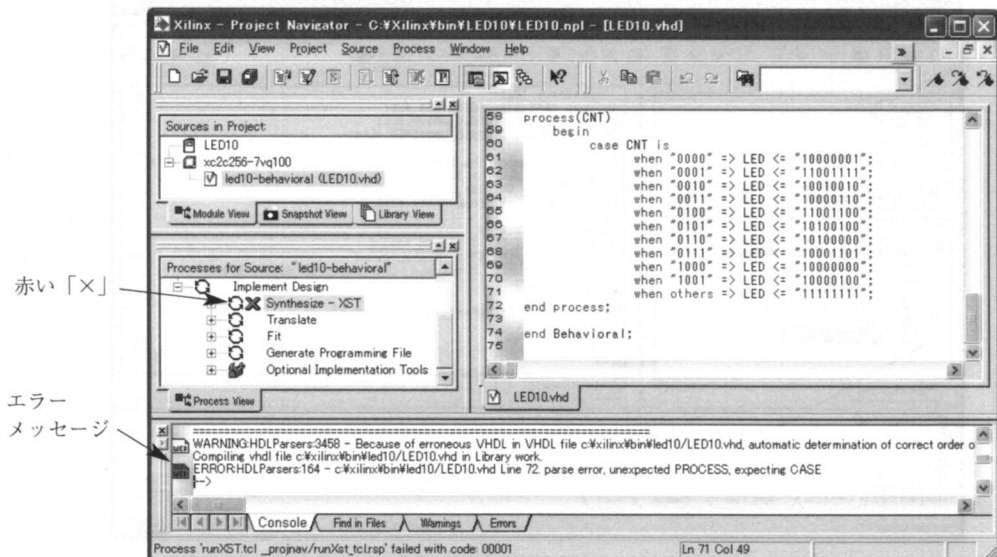



図 2.25　論理合成エラー

エラーメッセージの部分をダブルクリックすると，コード入力用ウインドウ内のエラー箇所にジャンプすることができます．メッセージを参考にしてエラーを修正し，「上書き保存」を行ってから，再び論理合成を試みます．図 2.25 では，72 行目に「end case;」が抜けていることがエラーの原因です．また，エラーほど重要でないが，問題点（WARNING）のある場合には，黄色い「！」マークが表示されます（図 2.26）．この場合は，メッセージを確認して，問題がないかどうか検討しましょう．

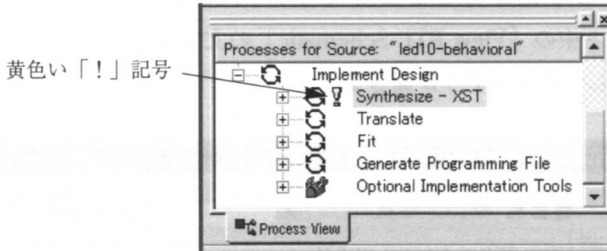



図 2.26　「WARNING」

論理合成が終了すれば，続いて配置配線に進むことができます．しかし，先へ進む前に，論理合成によって作成された情報のいくつかをみてみましょう．

① Synthesis Report

図 2.27 に示すように，［Process View］ウインドウの「Synthesize-XST」の左側をクリックして「−」記号にし，下層リストを表示します．そして，リスト中の「View Synthesis Report」の文字部分をダブルクリックすると，論理合成レポートが表示されます．

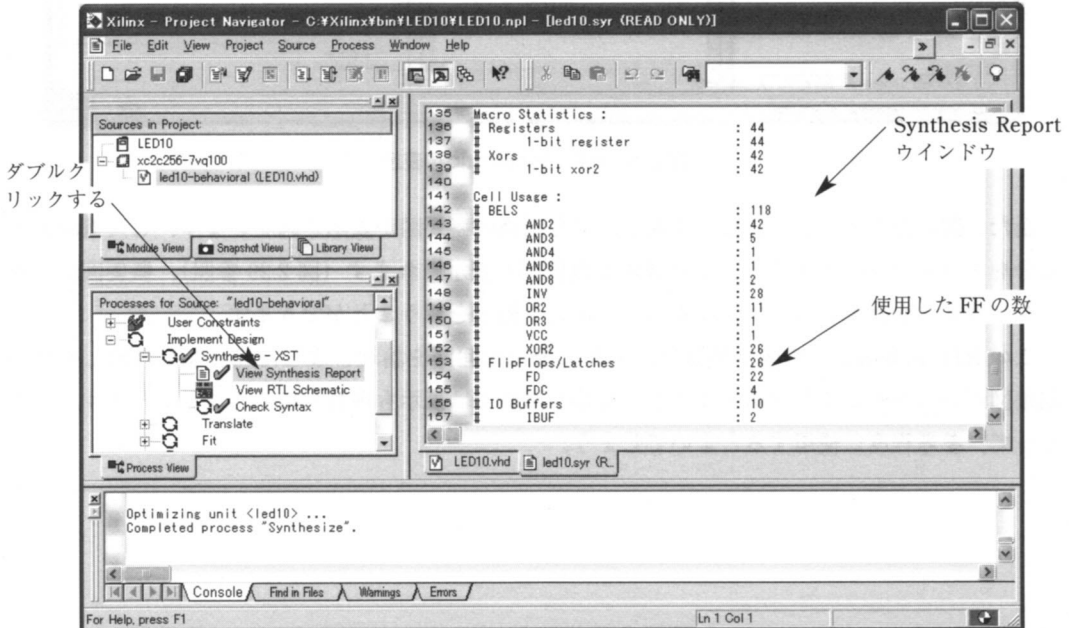



図 2.27　Synthesis Report の表示

Synthesis Reportウインドウには，論理合成の結果が表示されています．たとえば，図2.27のSynthesis Reportには，使用したFFの数が，22個と4個の計26個と表示されています．これは，実習回路で分周回路に22ビットカウンタ，10進カウンタ回路に4ビットカウンタを使用したからです．

② RTL Schematic

論理合成した回路を図として見ることができます．「Process View」ウインドウの「Synthesize-XST」の下層リスト中の「View RTL Schematic」の文字部分をダブルクリックすると，図2.28に示すように，新しいウインドウが開きます．

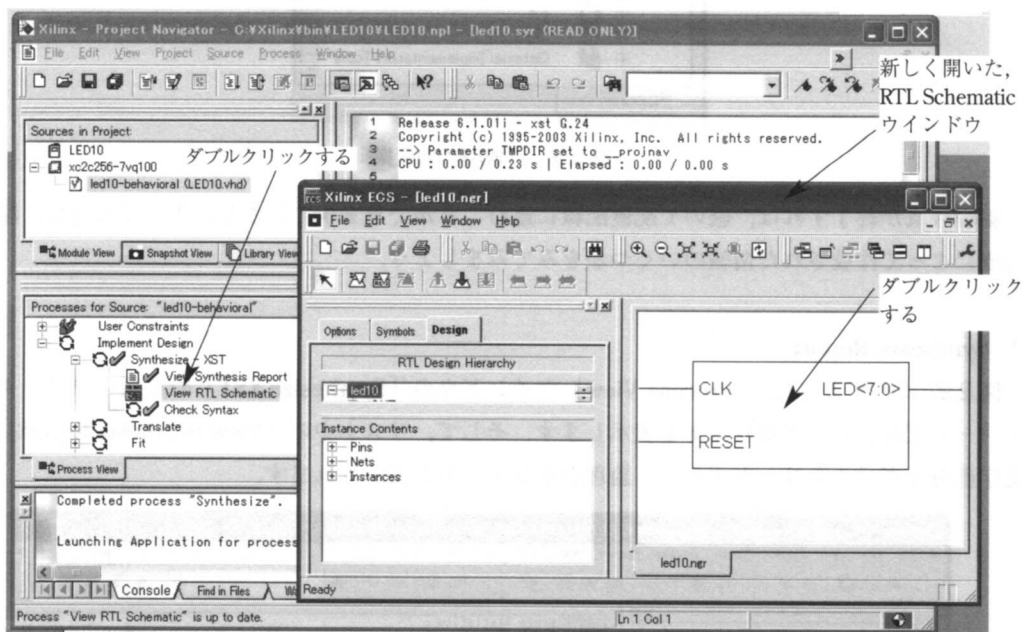



図2.28　RTL Schematicの表示

新たに開いたウインドウには，作成した実習回路の概要図が表示されています．この概略図の部分をダブルクリックすると，より詳細な概要図が表示されます（図2.29参照）．概要図は，ウインドウにおかれているボタンによって，拡大や縮小を行うことができます．

このRTL Schematicの表示機能は，非常に簡単な回路を論理合成した場合なら理解しやすい回路図が表示されることがありますが，複雑な回路では表示も理解しにくくなってきます．したがって，参考程度に使用するとよいでしょう．

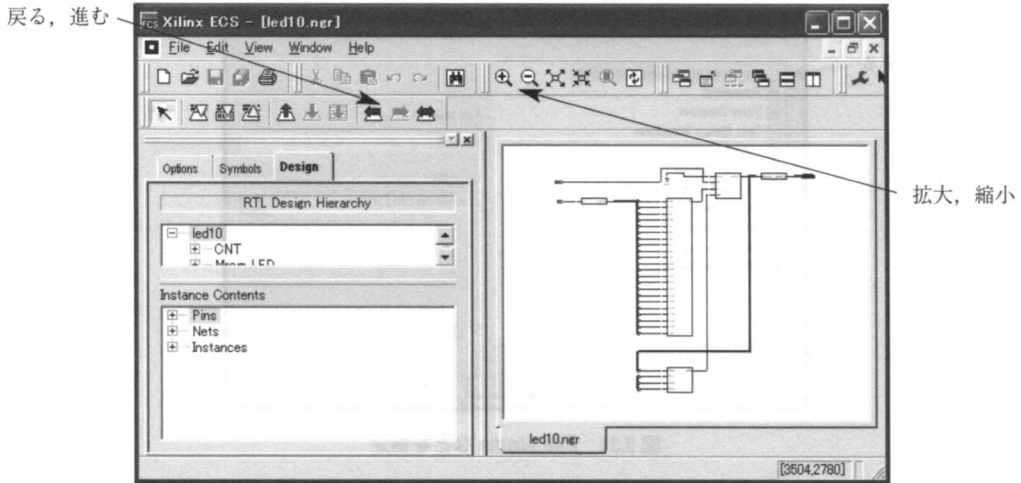



図 2.29　より詳細な概要図

▶2.2.4　配置配線

論理合成が終了したので，引き続いて配置配線を行います．

① ピン割り当てファイルの準備

配置配線を行うためにピンの指定を行いますが，まずピンの指定情報を記録するファイルを作成します．図 2.30 に示すように，メニューバーの「Project」→「New Source」を選択します．

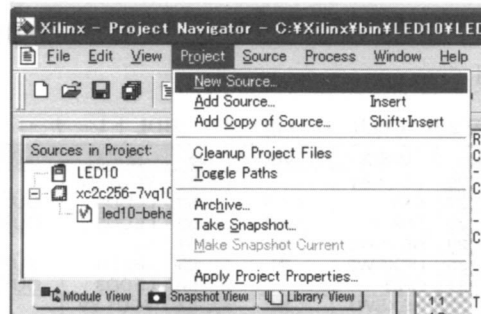



図 2.30　「New Source」を選択

すると図 2.31 に示すウインドウが表示されますので，左の一覧から「Implementation Constraints File」をクリックし，「File Name」欄に「LED10」と入力します．「次へ」ボタンをクリックして先へ進むと，いくつかの確認用ウインドウなどが表示されますので，そのまま「次へ」や「完了」ボタンをクリックして先へ進みます．すると，図 2.32 に示すように，「Module View」ウインドウに「LED10.ucf」というファイル名が表示されるはずです（この作業は，142 ページで説明するように省略することもできます）．

図2.31　File Nameなどを指定

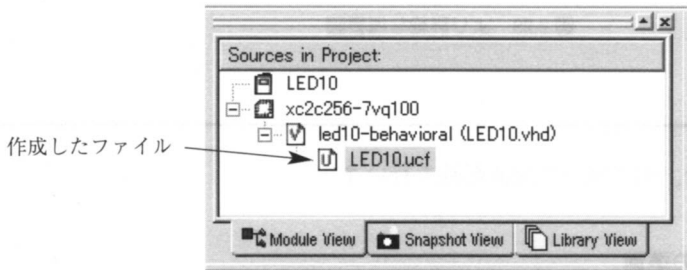



図2.32　作成した「LED10.ucf」

② ピン割り当ての確認

使用するHDLトレーナーに搭載されているCPLD「XC2C256-7VQ100C」のピン割り当ては，表2.5のようになっています．

表2.5　CPLDのピン割り当て（HDLトレーナー）

No.	接続先	I/O	No.	接続先	I/O	No.	接続先	I/O	No.	接続先	I/O
1	DEB0	INOUT	26	VCC1.8V		51	VCC3.3V		76	LEDB7	OUT
2	DEB1	INOUT	27	EXT	OUT	52	LEDD6	OUT	77	LEDB6	OUT
3	DEB2	INOUT	28	SPKOUT	OUT	53	LEDD5	OUT	78	LEDB5	OUT
4	DEB3	INOUT	29	HEXSW3	IN	54	NC		79	LEDB4	OUT
5	VAUX	IN	30	HEXSW2	IN	55	LEDD4	OUT	80	LEDB3	OUT
6	DEB4	INOUT	31	GND		56	LEDD3	OUT	81	LEDB2	OUT
7	DEB5	INOUT	32	HEXSW1	IN	57	VCC1.8V		82	LEDB1	OUT
8	DEB6	INOUT	33	HEXSW0	IN	58	LEDD2	OUT	83	TDO	OUT
9	DEB7	INOUT	34	SW1	IN	59	NC		84	GND	
10	DEB8	INOUT	35	SW2	IN	60	LEDD1	OUT	85	LEDB0	OUT
11	DEB9	INOUT	36	SW3	IN	61	LEDD0	OUT	86	LEDA7	OUT
12	DEB10	INOUT	37	SW4	IN	62	GND		87	LEDA6	OUT
13	DEB11	INOUT	38	VCC3.3V		63	NC		88	VCC3.3V	
14	DEB12	INOUT	39	AIN	INOUT	64	LEDC7	OUT	89	LEDA5	OUT
15	DEB13	INOUT	40	BZOUT	OUT	65	NC		90	LEDA4	OUT
16	DEB14	INOUT	41	IRXD	IN	66	NC		91	LEDA3	OUT
17	DEB15	INOUT	42	ITXD	OUT	67	LEDC6	OUT	92	LEDA2	OUT
18	NC		43	ICTS	OUT	68	LEDC5	OUT	93	NC	
19	NC		44	NC		69	GND		94	LEDA1	OUT
20	VCC3.3V		45	TDI	IN	70	LEDC4	OUT	95	NC	
21	GND		46	NC		71	LEDC3	OUT	96	NC	
22	CLK1	IN	47	TMS	IN	72	LEDC2	OUT	97	LEDA0	OUT
23	CLK2	INOUT	48	TCK	IN	73	LEDC1	OUT	98	VCC3.3V	
24	CLK2	INOUT	49	IRTS	IN	74	LEDC0	OUT	99	nRESET	IN
25	GND		50	LDOD7	OUT	75	GND		100	GND	

これより，実習回路では，表2.6に示す入出力ピンを使用することにします．

表2.6 実習回路で使用する入出力ピン

<table>
<tr><th colspan="2">記 号</th><th>ピン番号</th><th>備 考</th></tr>
<tr><td rowspan="2">入力</td><td>CLK</td><td>22</td><td>セラロック 4 MHz</td></tr>
<tr><td>RESET</td><td>34</td><td>プッシュスイッチ SW1</td></tr>
<tr><td rowspan="8">出力</td><td>LED 0</td><td>61</td><td rowspan="8">7セグメント LED
LED 4</td></tr>
<tr><td>LED 1</td><td>60</td></tr>
<tr><td>LED 2</td><td>58</td></tr>
<tr><td>LED 3</td><td>56</td></tr>
<tr><td>LED 4</td><td>55</td></tr>
<tr><td>LED 5</td><td>53</td></tr>
<tr><td>LED 6</td><td>52</td></tr>
<tr><td>LED 7</td><td>50</td></tr>
</table>

③ ピン割り当て作業

図2.33に示すように，「Process View」ウインドウの「User Constraints」の下層メニューを表示し，「Assign Package Pins」をダブルクリックします．

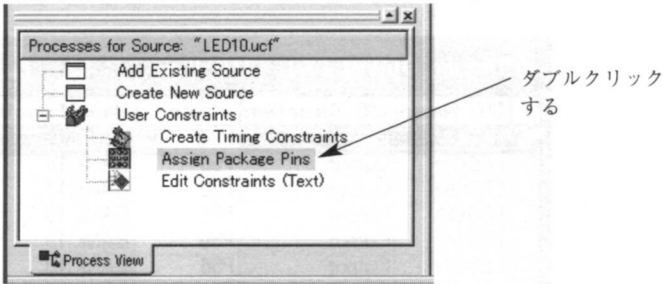



図2.33 「Assign Package Pins」をダブルクリック

すると，図2.34に示すようなウインドウが開きます．

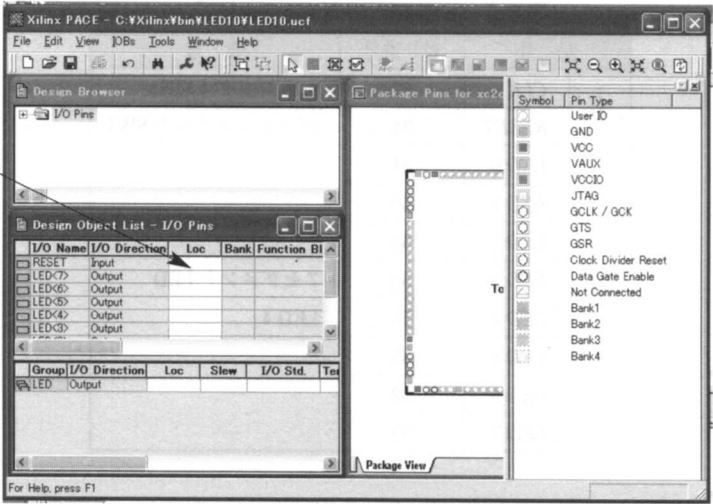



図2.34　ピン割り当てウインドウ

左側の「Design Object List − I/O Pins」ウインドウ（図2.35参照）の「Loc」欄に「RESET」→「P34」のようにピン番号を入力していきます．ピン番号は，表2.6を参照してください．

Design Object List - I/O Pins

I/O Name	I/O Direction	Loc	Bank	Function Block
RESET	Input	P34	BANK 8	
LED<7>	Output	P50	BANK 14	
LED<6>	Output	P52	BANK 14	
LED<5>	Output	P53	BANK 13	
LED<4>	Output	P55	BANK 13	
LED<3>	Output	P56	BANK 13	
LED<2>	Output	P58	BANK 15	
LED<1>	Output	P60	BANK 15	
LED<0>	Output	P61	BANK 15	
CLK	Input	P22	BANK 5	

Group	I/O Direction	Loc	Slew	I/O Std.	Te
LED	Output				

図2.35　ピン番号の入力

ピン番号の入力を終えたら，「上書き保存」ボタンをクリックしてファイルを保存し，ピン割り当てウインドウを閉じておきます（図2.36参照）．

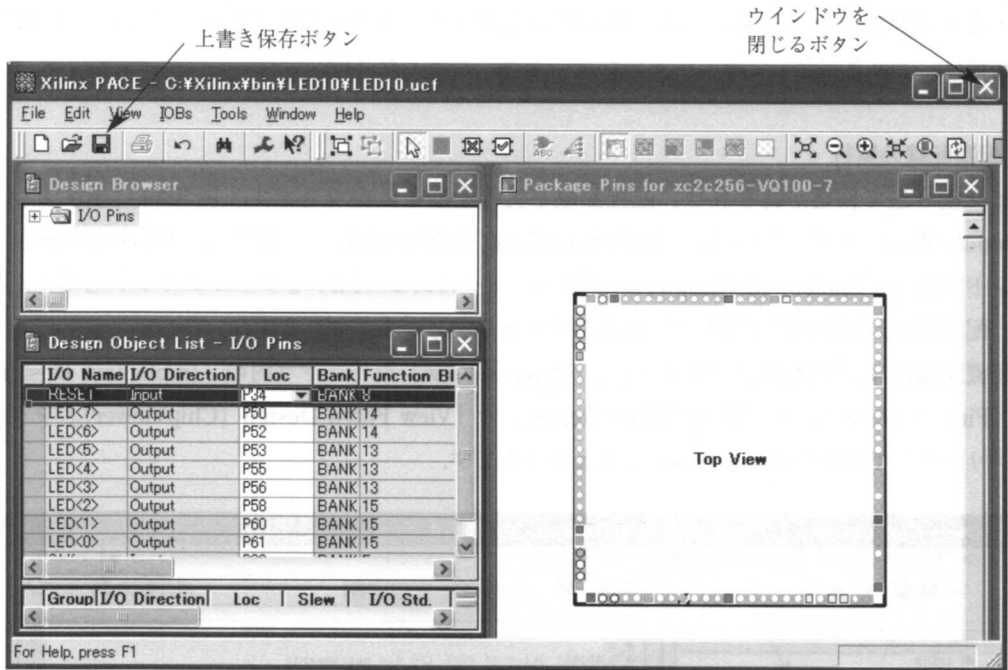



図2.36　ファイルを保存してウインドウを閉じる

④ テキストエディタによる確認

ピン割り当て情報を記録したファイル「LED10.ucf」を表示して，先ほどの設定を確認しましょう．ファイル「LED10.ucf」の表示は，「Process View」ウインドウの「Edit Constraints (Text)」をダブルクリックするか，右側ウインドウの「LED10.ucf」タグをクリックします（図2.37参照）．

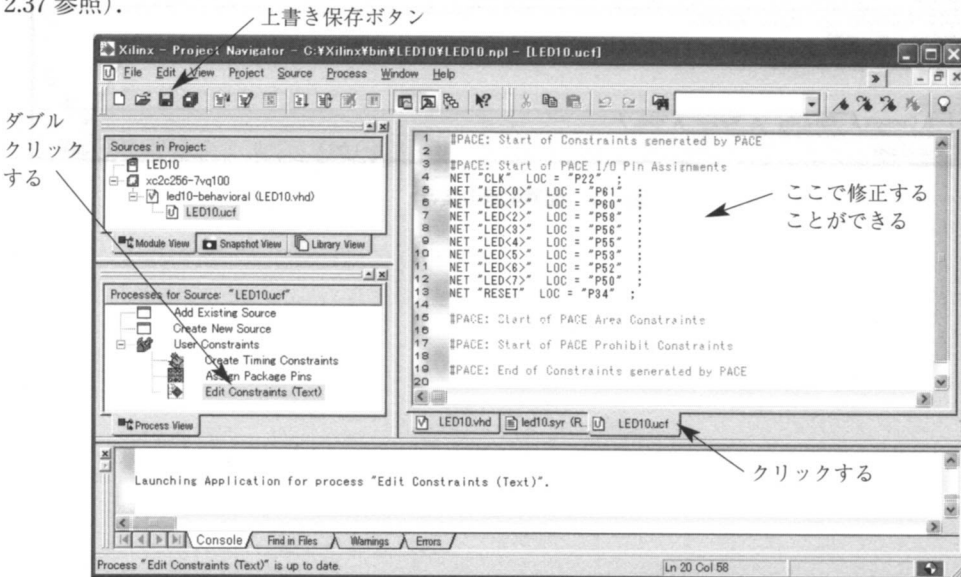



図2.37　「LED10.ucf」の内容を表示

入力ミスが見つかった場合には，図2.37の右側ウインドウ内で修正を行うことができます．修正後は，上書き保存しておくのを忘れないようにしてください．

⑤ 配置配線の実行

ピン割り当て作業が終われば，配置配線を実行することができます．図2.38に示すように「Module View」ウインドウ内の「led10-behavioral（LED10.vhd）」を選択し，「Process View」ウインドウ内の「Implement Design」の下層メニューにある「Fit」をダブルクリックしてください．配置配線が無事に終了すれば，緑色のチェック記号が表示されます．

配置配線後は，図2.39に示すように，「Process View」ウインドウ内の「Implement Design」→「Fit」の下層メニューにある「Fitter Report」や「View Fitted Design（ChipViewer）」をダブルクリックして各種の情報を表示することができます．

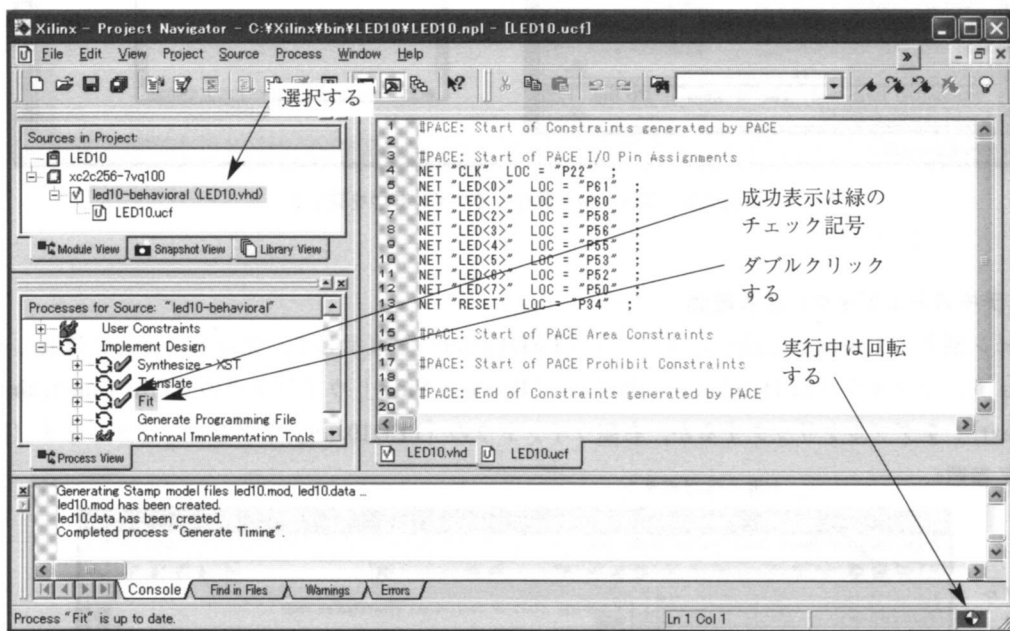



図2.38　配置配線の実行

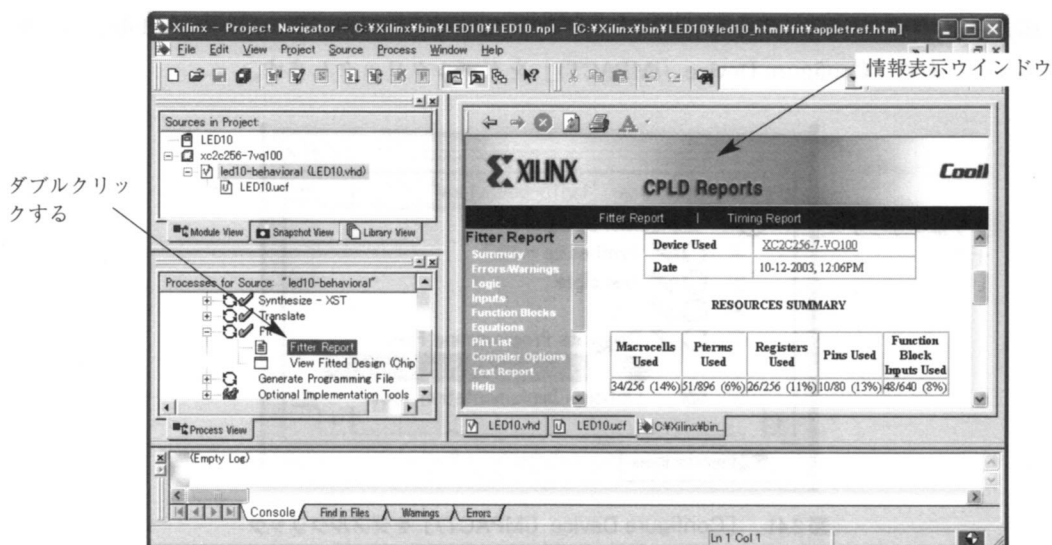



図 2.39 配置配線後の情報表示

▶2.2.5 ダウンロード

続いて評価ボードにダウンロードを行うためのファイルを作成します．図 2.40 に示すように，「Process View」ウインドウ内の「Generate Programming File」をダブルクリックします．成功を示す緑のチェック記号が表示されたら終了です．

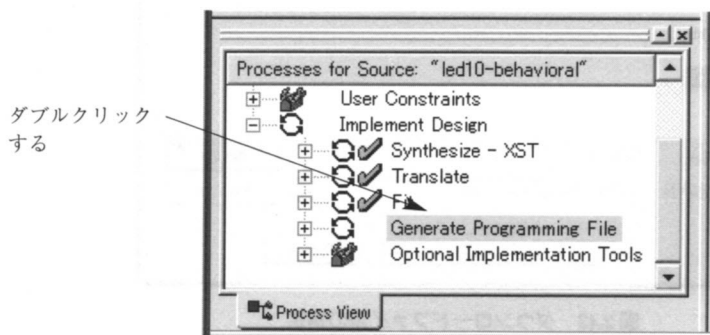



図 2.40 ダウンロード用ファイルを作成

評価ボードをパソコンに接続し，AC アダプタで電源を供給します．詳しくは，第 7 章 196 ページを参照してください．

図2.41に示すように，「Process View」ウインドウ内の「Generate Programming File」の下層メニューにある「Configure Device（iMPACT）」をダブルクリックします．

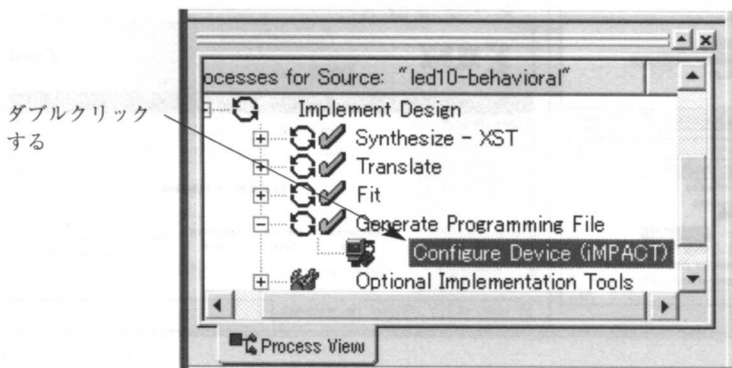



図2.41　「Configure Device（iMPACT）」をダブルクリック

いくつかのウインドウが表示されますが，ここではすべてそのまま次へ進みます．そして，図2.42に示すウインドウが表示されたら，ダウンロードファイル「led10.jed」を選択して「開く」ボタンをクリックします．

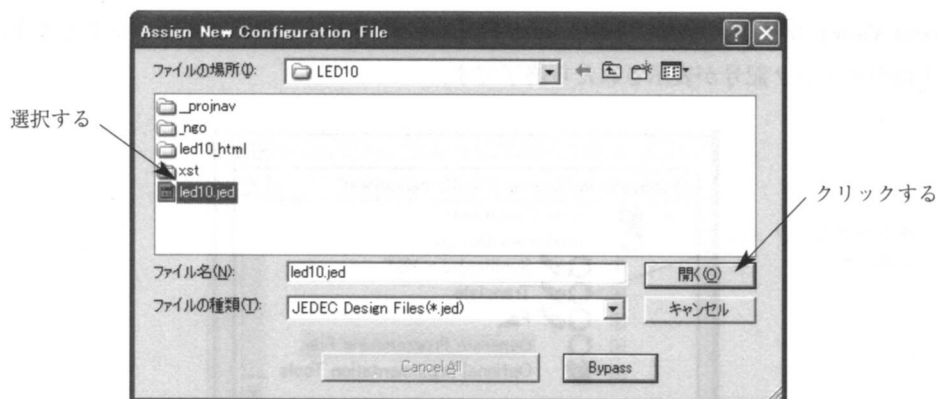



図2.42　ダウンロードファイルの指定

図2.43に示すように，CPLDの図を右クリックして表示されるメニューから「Program」を選択します．

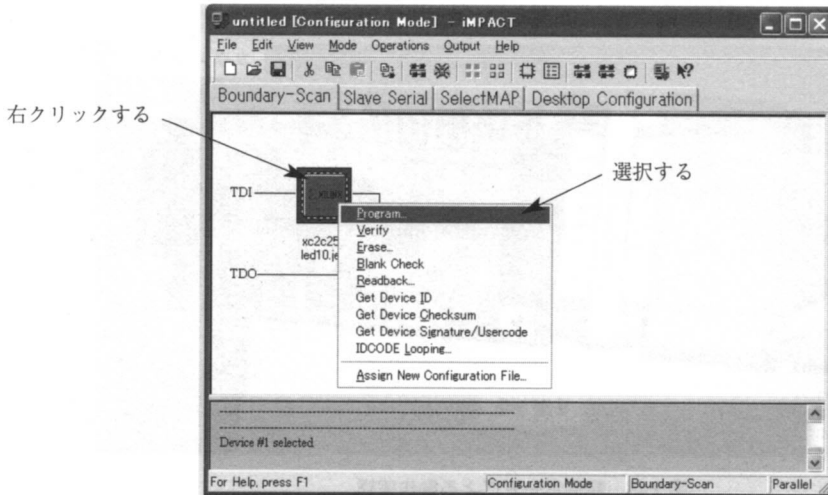



図2.43 「Program」を選択

続いて現れるウインドウでは，そのまま「OK」ボタンをクリックします．すると，CPLDへのダウンロードが始まります．その後，図2.44に示すように，ウインドウに「Programming Succeeded」と表示されればダウンロードの成功です．

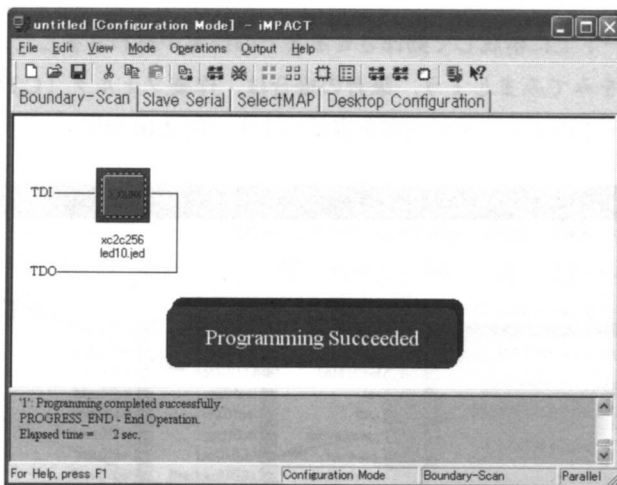



図2.44 ダウンロードの終了

▶2.2.6 評価ボードによる動作確認

評価ボードの動作を確認してみましょう．一番右側の7セグメントLEDが0から9までの表示を正しく繰り返していますか．また，一番左側のプッシュスイッチを押すとリセットがかかり，

7セグメントLEDの表示が0から再スタートするでしょうか（図2.45参照）．もしも，エラーのある場合には，記述したコードやピン割り当てなどに間違いがないかどうか点検しましょう．間違いに応じて，論理合成や配置配線，ダウンロードなどの操作を繰り返す必要があります．

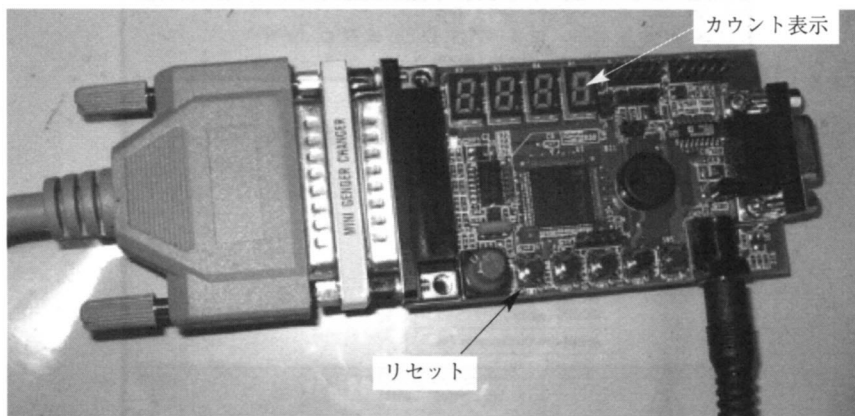



図2.45　評価ボードによる動作確認

正しい動作が確認できたら，ウインドウ右上の ✕ ボタンをクリックしてISE WebPACKを終了します．

▶2.2.7　作成された各種ファイル

実習回路を評価ボード上に構成して動作させるまでの作業が終わりました．ここまでの作業で作成されたファイルをみてみましょう．筆者の場合は，作業フォルダ「C:¥Xilinx¥bin¥LED10」を指定しましたので，このフォルダの内容を表示します（図2.46参照）．

図2.46　C:¥Xilinx¥bin¥LED10

数多くのファイルが作成されていますが，主要ないくつかのファイルについて説明しておきましょう．

①「LED10.npl」 → 作業を統合して管理するファイルです．このファイルのアイコンをダブルクリックすれば，ISE WebPACK が起動し，実習回路作成の作業を再開できます．

②「LED10.vhd」 → VHDL のコードが記録されたテキストファイルです．

③「LED10.ucf」 → CPLD のピン割り当て情報が記録されたファイルです．

④「led10.jed」 → 評価ボードへダウンロードするファイルです．

図 2.46 にあるすべてのファイル容量は，およそ 2 MB です．したがって，たとえば作成したデータを他のパソコンへ移動する場合など，すべてのファイルをコピーしていたのではたいへんです．このような場合は，「LED10.vhd」だけをコピーして，コピー先のパソコンで再度 ISE WebPACK 6.1i による作業を行えばよいでしょう．「LED10.vhd」は，VHDL コードを記述したテキストファイルですから，ファイル容量も小さく（この場合は 1.6 kB），ワードパッドなどのエディタを使用して編集することも可能です．

▶ 演習問題 2

2.1　ターゲットデバイスの選定条件を挙げなさい．

2.2　つぎの条件を満たす CPLD を選定したい．29 ページの表 2.1，30 ページの表 2.2 を参考にして適切なザイリンクス社の CPLD を答えなさい．

① 回路には，およそ 80 個のフリップフロップを使用する．

② 動作電圧は，3.3 V とする．

③ 入出力ピンとして，70 本を使用する．

④ パッケージ形状は不問である．

⑤ 自動車に搭載するため，周囲の温度条件は厳しい．

2.3　CPLD/FPGA の動作速度と消費電力の関係について簡単に説明しなさい．

2.4　実習回路（34 ページ図 2.5）において，各回路の働きを簡単に説明しなさい．

① 10 進数カウンタ回路

② 分周回路

③ デコーダ回路

2.5　実習回路（34 ページ図 2.5）と ISE WebPACK による入出力ピンの設定（40 ページ図 2.17）を関連づけて説明しなさい．

2.6　実習回路を構成した後にできるファイル「LED10.vhd」はどのようなファイルか，簡単に説明しなさい．

2.7　実習回路の一部を変更するため，ISE WebPACK を起動したい．どのようにすればよいか説明しなさい．

2.8　実習回路を動作させた場合，評価ボードの右から 2 番目の 7 セグメント LED を表示用，左から 2 個目のプッシュスイッチをリセット用にしたい．

① ISE WebPACK を使用した場合，どの手順から作業を行えばよいか．

② 使用する入出力ピンについて表 2.7 を完成させなさい．

③ 評価ボードで変更後の動作を確認しなさい．

表 2.7　変更後の入出力ピン

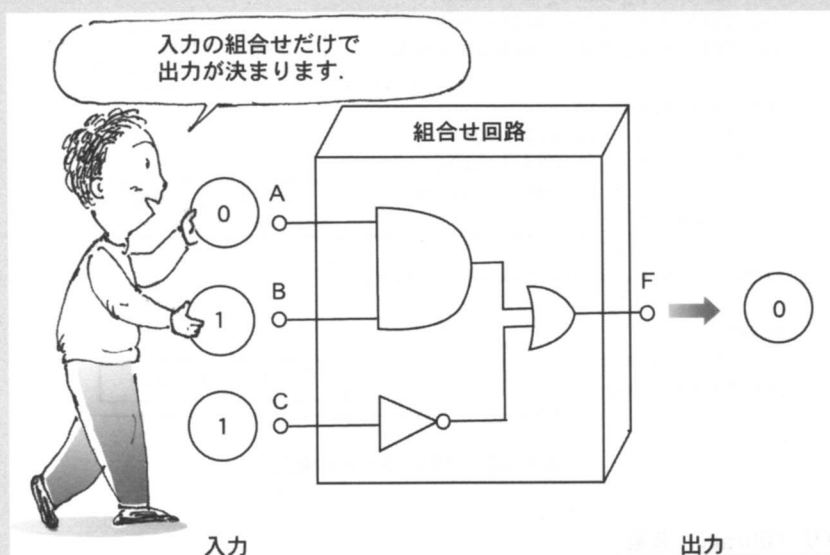
記号		ピン番号	備考
入力	*CLK*		セラロック 4 MHz
	RESET		プッシュスイッチ SW2
出力	*LED* 0		7 セグメント LED LED 3
	LED 1		
	LED 2		
	LED 3		
	LED 4		
	LED 5		
	LED 6		
	LED 7		

第 3 章 組合せ回路の設計

ディジタル回路は，組合せ回路と順序回路に大別できます．組合せ回路は入力の組合せだけで出力が決まりますが，順序回路では入力の組合せに加えて回路の内部状態がどのようになっているかで出力が決まります．

この章では，VHDL を用いて組合せ回路を設計する方法を学習しましょう．章の前半では，VHDL 記述の基本的な構成や文法などについて説明します．その後，組合せ回路として，加算回路や減算回路，デコーダやエンコーダなどの設計法を説明します．

この章で取り上げる例題は，ぜひとも実際に評価ボードへダウンロードして動作を確認してください．そして，各自で考えた回路変更などを行う実習をしてください．

第3章

3.1 VHDLの書き方

ここでは，VHDLの基本的な構成について説明します．VHDLのコードは，ライブラリ宣言，エンティティ宣言，アーキテクチャ宣言などから構成されています．各部の意味や記述のパターンを理解しましょう．

▶3.1.1 基本構成

図3.1に示すAND回路をVHDLで記述するとリスト3.1のようになります．リスト3.1に示すように，VHDLの基本的なコードは，ライブラリ宣言，エンティティ宣言，アーキテクチャ宣言の3ブロックから構成されます．各ブロックの内容をみていきましょう．

A	*B*	*F*
0	0	0
0	1	0
1	0	0
1	1	1

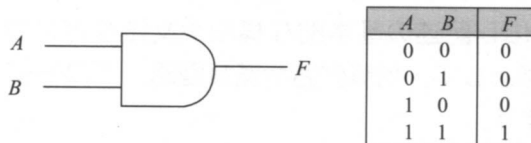



図3.1　AND回路

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;          ライブラリ宣言

entity sample1 is
    Port ( A : in std_logic;
           B : in std_logic;
           F : out std_logic);
end sample1;                               エンティティ宣言

architecture Behavioral of sample1 is

begin
     F <= A and B ;
end Behavioral;                            アーキテクチャ宣言
```

リスト3.1　VHDLコードの構成

① ライブラリ（library）宣言

ライブラリ宣言は，IEEEで標準化された各種のデータ型や演算子などを使用するために必要なライブラリとパッケージを指定します．リスト3.2は，リスト3.1のライブラリ宣言を抜き出

したものです．当面は，パターンとしてリスト3.2のように記述しておけばよいでしょう．図3.2に，ライブラリ宣言の書式を示します．ザイリンクス社の開発ツールISE WebPACKでは，ライブラリ宣言を自動的に記述してくれます．

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

リスト3.2 ライブラリ宣言

```
library [ライブラリ名] ;
use [ライブラリ名] . [パッケージ名] ;
```

図3.2 ライブラリ宣言の書式

② エンティティ（entity）宣言

エンティティ宣言は，入力と出力の設定を行います．図3.3に示すように，ディジタル回路の入出力端子を宣言するのです．

図3.3 入出力端子の宣言

具体的には，エンティティ宣言中のポート（port）宣言によって，入出力端子の設定を行います．リスト3.3は，リスト3.1のエンティティ宣言を抜き出したものです．図3.4に，エンティティ宣言の書式を示します．エンティティ名は，他のファイル名（プロジェクト名）と重複しないようにしましょう．ポート名は，入出力端子の記号を表します．VHDLでは，大文字と小文字の区別はありません．

```
entity sample1 is
    Port ( A : in std_logic;
           B : in std_logic;
           F : out std_logic);
end sample1;
```

リスト3.3 エンティティ宣言

図3.4 エンティティ宣言の書式

・モード型

表3.1に示す三種類の記号で信号の方向を表します（図3.5参照）．bufferというモード型もありますが，ザイリンクス社のデバイスではサポートされていません．

・データ型

「std_logic」は，1ビットの信号を表します．複数ビットのデータをまとめて扱いたい場合には，「std_logic_vector（　）」を使います．

表 3.1　モード型

記号	方向
in	入力
out	出力
inout	双方向

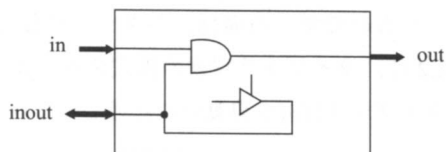



図 3.5　モード型

例）8ビットのデータをまとめて扱う場合：「std_logic_vector（7 downto 0）」
データ型については，後で詳しく説明します．

③ アーキテクチャ（architecture）宣言

アーキテクチャ宣言は，ディジタル回路の機能を記述するブロックです（図3.6参照）．

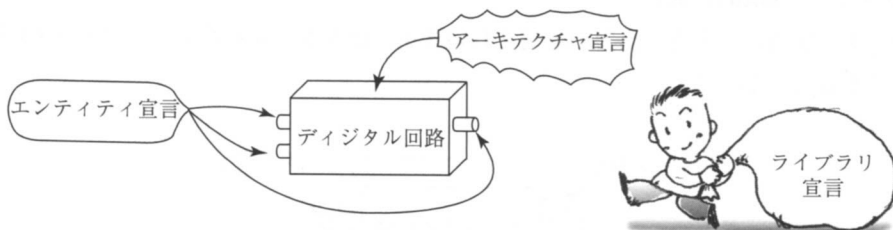



図 3.6　アーキテクチャ宣言は機能の記述

リスト3.4は，リスト3.1のアーキテクチャ宣言を抜き出したものです．図3.7に，アーキテクチャ宣言の書式を示します．

```
architecture Behavioral of sample1 is

begin
     F <= A and B ;
end Behavioral;
```

リスト 3.4　アーキテクチャ宣言

```
architecture [アーキテクチャ名] of [エンティティ名] is
      ( 信号宣言部 )
begin
      ( 機能宣言部 )
end [アーキテクチャ名] ;
```

図 3.7　アーキテクチャ宣言の書式

アーキテクチャ名は，他のファイル名と重複しても支障がないため，ここでは「Behavioral」を使用しました．この他，エンティティ名と関連付けしやすい名前にすることも多いようです．たとえば，エンティティ名が「sample1」ならば，アーキテクチャ名を「a_sample1」などのようにします．このコードでは，信号宣言部を使用していませんが，これについては後で説明します．機能宣言部は，「begin」と「end」に挟まれた領域で，ここにディジタル回路の論理機能を記述します．リスト3.4の機能宣言部は，論理演算子「and」を用いて，図3.8に示すような2入力AND回路の機能を記述しています．

演算結果の代入（出力）には「<=」記号を使用します．論理演算子には，表3.2に示す種類があります．演算子には，記述の順序による優先順位がありませんので，複数の演算子を同じ文

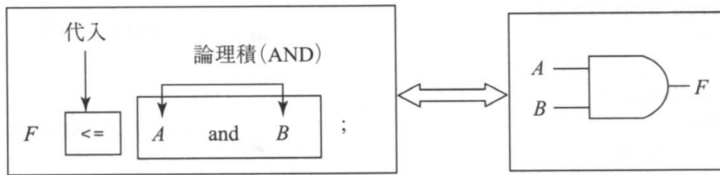



図 3.8　機能宣言部

で使用する場合には，カッコ（　）を使用して優先順位を定める必要があります．

表 3.2　論理演算子

演算子	機能
and	論理積
or	論理和
not	論理否定
nand	否定論理積
nor	否定論理和
xor	排他的論理和
xnor	否定排他的論理和

各宣言部の働きが理解できたならば，リスト 3.1 に示したコードを評価ボードで動作させる実習を行いましょう．HDL トレーナーを使用した場合のピン割り当ての例を表 3.3 に示します．2 個のプッシュスイッチ SW1 と SW2 を入力 A，B に，出力 F を 7 セグメント LED（LED4 のセグメント g）に割り当てます．

表 3.3　リスト 3.1 のピン割り当て（HDL トレーナー）

記号		ピン番号	備考
入力	A	35	SW1
入力	B	34	SW2
出力	F	61	LED4 の g

第 2 章 35 ページで説明した手順にしたがって，コードの記述，論理合成，配置配線，ダウンロードと操作していきます．論理合成が終了した時点で，ISE WebPACK 画面の「Process View」ウインドウ内の「Synthesize-XST」→「View Synthesis Report」を選択すると（45 ページ図 2.27 参照）合成された回路の情報をみることができます（図 3.9）．

また，「Synthesize-XST」→「View RTL Schematic」を選択すると（46 ページ図 3.28 参照）合成された回路図を見ることができます（図 3.10）．

ダウンロードまで終了したら，プッシュスイッチを押して AND 回路として正しく動作しているかどうかを確認しましょう．ただし，HDL トレーナーではプッシュスイッチや 7 セグメント LED などは負論理（ON ＝信号 0）で動作することに注意しましょう．図 3.11 に示すように，

図 3.9　Synthesis Report の一部

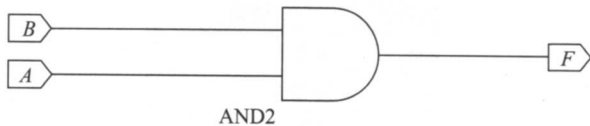



図 3.10　論理合成された回路（RTL Schematic）

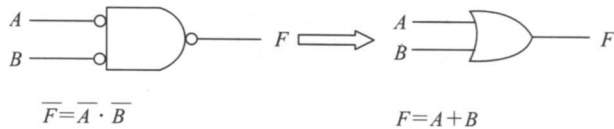



図 3.11　ド・モルガンの定理

AND 回路の入出力を負論理にすると正論理の OR 回路として動作します（ド・モルガンの定理）．

▶3.1.2　データ型

VHDL には，表 3.4 に示すような多くのデータ型がありますが，最もよく使用するのは，「std_logic」と「std_logic_vector」です．

「std_logic」では，0 と 1 の 2 値以外に，ハイインピーダンス Z や不定値 X などの 1 ビットデータを扱うことができます．そして，これらのデータを複数ビットにまとめて扱いたい場合には，「std_logic_vector」を用います．具体例で確認しましょう．

表 3.4　データ型（データタイプ）

データ型	説明	データ型	説明
std_logic	0, 1, Z, X などの論理値	boolean	TRUE，FALSE の論理値
std_logic_vector	std_logic のベクトル型	integer	整数値（32 ビット）
bit	0,1 の論理値	character	文字（ASCII）
bit_vector	bit のベクトル型	string	character のベクトル型

図 3.12 に示すような，4 ビットの入力信号 $A0$ ～ $A3$ を受け取り，4 ビットの出力信号 $F0$ ～ $F3$ を出す回路を考えます．ここでは，データ型の理解が目的なので，入力信号がそのまま出力される回路とします．

この回路をデータ型「std_logic」を使用して記述すると，リスト 3.5 のようになります．リスト 3.5 のエンティティ宣言は，リスト 3.6 のように書くことも可能ですが，リスト 3.6 で使用しているデータ型は同じ「std_logic」です．したがって，アーキテクチャ宣言では，どちらのリストでも 1 ビット単位の代入処理を記述しなければなりません．

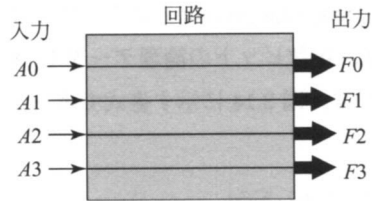



図 3.12　多入力，多出力回路

リスト 3.5 を論理合成した結果の回路図を図 3.13 に示します．リスト 3.6 を論理合成した結果も同じになります．

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity sample2 is
    Port ( A0: in std_logic;
           A1: in std_logic;
           A2: in std_logic;
           A3: in std_logic;
           F0: out std_logic;
           F1: out std_logic;
           F2: out std_logic;
           F3: out std_logic);
end sample2;

architecture Behavioral of sample2 is

begin
        F0 <= A0;
        F1 <= A1;
        F2 <= A2;
        F3 <= A3;
end Behavioral;
```

リスト 3.5　「std_logic」を用いたコード 1

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity sample3 is
    Port ( A0,A1,A2,A3: in std_logic;
           F0,F1,F2,F3: out
std_logic);
end sample3;

architecture Behavioral of sample3 is

begin
        F0 <= A0;
        F1 <= A1;
        F2 <= A2;
        F3 <= A3;
end Behavioral;
```

リスト 3.6　「std_logic」を用いたコード 2

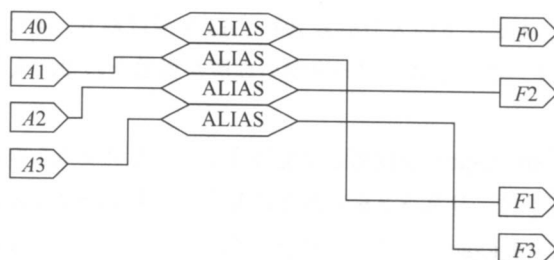



図 3.13　リスト 3.5 の回路図 (RTL Schematic)

つぎに，同じ回路をデータ型「std_logic_vector」によって記述しましょう．リスト 3.7 をみてください．「std_logic_vector」は，複数ビットの論理データをベクトルとしてまとめて扱うことができます．エンティティ宣言では，図 3.14 に示す書式でポートのビット幅と型を宣言します．

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity sample4 is
    Port ( A: in std_logic_vector(3 downto 0);
           F: out std_logic_vector(3 downto 0));
end sample4;

architecture Behavioral of sample4 is

begin
        F <= A;
end Behavioral;
```

一度に代入できる

リスト 3.7　「std_logic-vector」を用いたコード

図 3.14　「std_logic_vector」の書式

そして，同じデータ型で同じビット幅をもった信号であれば，「F <= A ;」のように一度にまとめて代入処理を行う記述が可能です．図 3.15 に，リスト 3.7 を論理合成した後の回路図を示します．

図 3.15　リスト 3.7 の回路図 (RTL Schematic)

▶3.1.3 signal文

図3.16に示すような内部信号*X*，*Y*をもった回路を考えましょう．

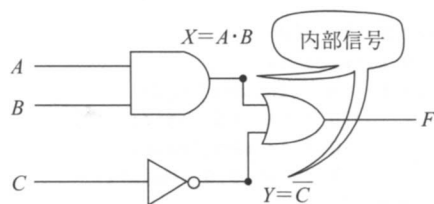



図3.16　内部信号をもった回路

この回路は，論理演算子を組合せてつぎのように記述することができます．

F <= (A and B) or (not C)；

しかし，回路によっては内部信号を考える必要が生じますので，ここではあえて内部信号*X*，*Y*を意識したコードの記述法を説明します．

61ページで学んだエンティティ宣言は，回路から外部への入出力端子を設定するものでした．一方，図3.17に示すように回路内部の信号は**signal**（シグナル）文で扱います．

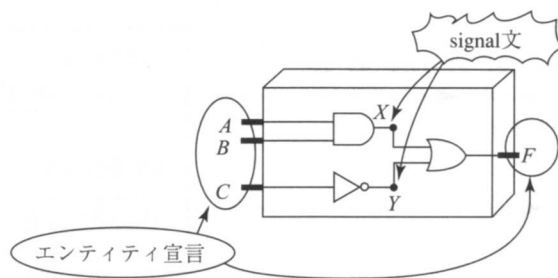



図3.17　内部信号はsignal文で扱う

リスト3.8に，signal文を用いて図3.16の回路を記述したコードを示します．VHDLの基本的なコードは，ライブラリ宣言，エンティティ宣言，アーキテクチャ宣言の三つのブロックから構成されていることを再確認してください．

内部信号は，signal文で信号名とデータ型を宣言してから使用します．宣言をする場所は，アーキテクチャ宣言の信号宣言部です（62ページ図3.7参照）．宣言を行った内部信号は，機能宣言部で他の入出力信号と同じように使用することができます

リスト3.9は，リスト3.8のアーキテクチャ宣言を抜き出したものです．図3.18に，signal文の書式を示しますので対応させて理解してください．

なお，リスト3.8中の「--」で始まる行はコメント文です．コメント文は，論理合成などの処理時に無視されますので，うまく使用してコードをわかりやすく記述するのに役立てるとよいでしょう．

第3章

```
-- ライブラリ宣言
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- エンティティ宣言
entity sample5 is
    Port ( A : in std_logic;
           B : in std_logic;
           C : in std_logic;
           F : out std_logic);
end sample5;

-- アーキテクチャ宣言
architecture Behavioral of sample5 is
           signal X:std_logic;
           signal Y:std_logic;
begin
           X <= A and B;
           Y <= not C;
           F <= X or Y;
end Behavioral;
```

リスト 3.8　内部信号を考えたコード

```
-- アーキテクチャ宣言
architecture Behavioral of sample5 is
    signal X:std_logic;
    signal Y:std_logic;
begin
    X <= A and B;
    Y <= not C;
    F <= X or Y;
end Behavioral;
```


リスト 3.9

図 3.18　signal 文の書式

リスト 3.8 を論理合成した後の回路図を図 3.19 に示します．

ここで，大切な考え方を説明します．リスト 3.9 のアーキテクチャ宣言の機能宣言部をみてください．信号 X，Y，F への代入処理が合計 3 回記述されています．注意すべき点は，これらの代入処理が同時に実行されることです．このことについて，もう少し詳しく説明しましょう．

リスト 3.10 は，C 言語のあるプログラムの一部を示したものです．

```
x = a * b ;
y = x ;
f = y + a  ;
```

上から順次実行される

リスト 3.10　C 言語プログラムの一部

このプログラムを実行すると，初めにリスト 3.10 の 1 行目が実行され，その後 2 行目，そして 3 行目へと処理が順次進んでいきます．たとえば，2 行目の実行には，すでに終了している 1 行目の結果が必要となります．つまり，プログラムは順次（シーケンシャル：sequential）処理

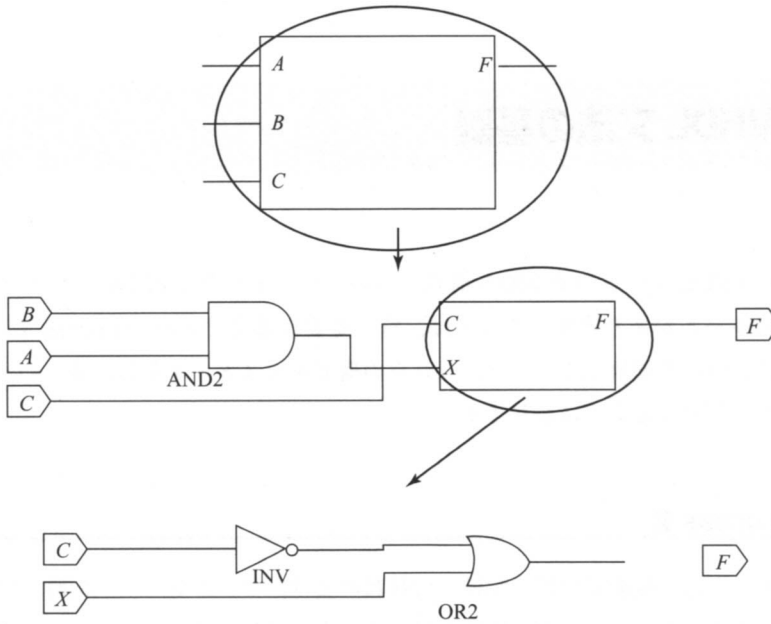
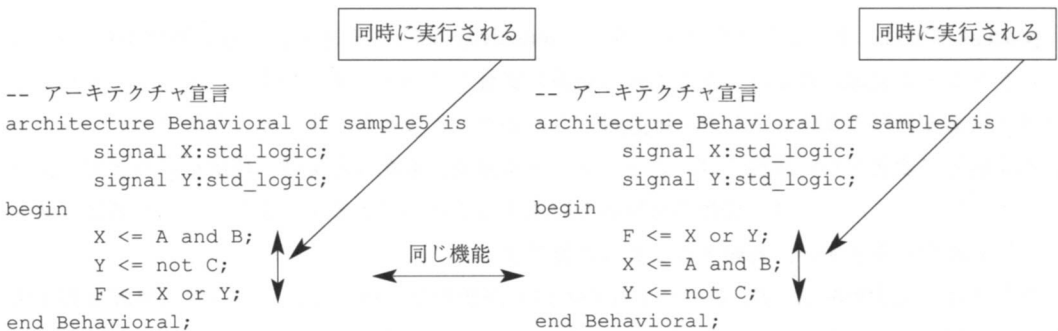



図 3.19 リスト 3.8 を論理合成した回路図（RTL Schematic）

されています．一方，リスト3.9のアーキテクチャ宣言の機能宣言部では，信号*X*，*Y*，*F*への代入処理はすべて同時（コンカレント：concurrent）に実行されるのです．つまり，リスト3.9の代入処理では記述の順序は意味をもちません．このため，たとえば，リスト3.11とリスト3.12は同じ機能を表します．

以上のようなことから，リスト3.11やリスト3.12で記述した代入文をコンカレント文とよぶこともあります．VHDLにおいて，シーケンシャルな処理を行わせたい場合には，後で学ぶ「プロセス文」を使用します．実際に論理合成を行い，リスト3.11とリスト3.12の回路図が同じになることを確認してください．

リスト 3.11 X,Y,F の順に記述

リスト 3.12 F,X,Y の順に記述

3.2 VHDL 文法の基礎

これまでに，VHDL コードの基本的な構成について学びました．VHDL のコードを記述する文法には多くのルールがあります．ここでは，特に重要な構文である，process 文，if 文，case 文などの使用法について理解しましょう．これらの構文をマスターすると，多くのディジタル回路を記述することができるようになります．

▶3.2.1　process 文

アーキテクチャ宣言の機能宣言部で記述した複数の文は，コンカレント（同時）に並列処理されることを前項で学びました．一方，VHDL コードにおいて，一般のプログラミング言語（C 言語など）と同じように，文をシーケンシャル（一つずつ順番）に実行させたい場合があります．このようなときには，図 3.20 に示す process（プロセス）文を使用します．

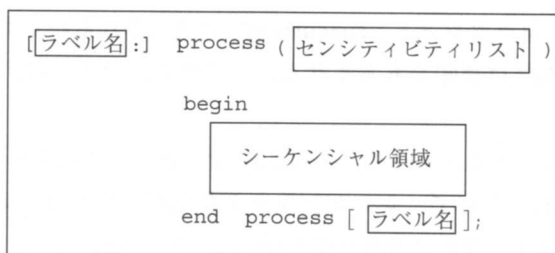



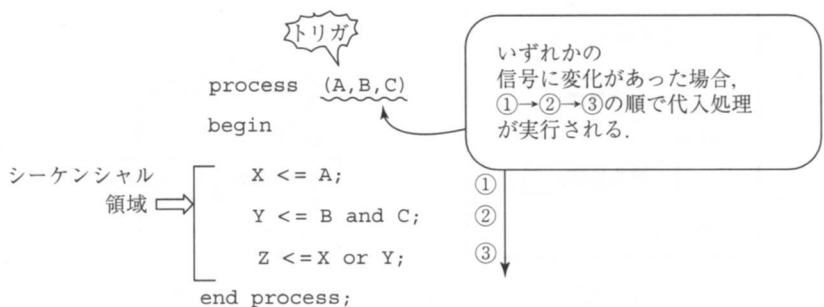
図 3.20　process 文の書式

process 文では，センシティビティリスト（sensitivity list）に記述した信号に変化が起こると，シーケンシャル領域に書かれた文を上から順番に実行していきます．つまり，センシティビティリストにある信号の変化がトリガ（trigger：きっかけ）となるのです．センシティビティリストにある信号に変化がない状態では，シーケンシャル領域に書かれた文は一切実行されません．センシティビティリストには，複数の信号名を記述することができます．また，ラベル名はコメントとして使用できますが，省略することも可能です．

たとえば，入力信号 A，B，C のいずれかの信号が変化した場合に，三つの代入処理を順次実行したい場合には，図 3.21 に示すような記述を行います．図 3.21 の process 文のトリガが有効になった場合にのみ，シーケンシャル領域に書かれた文が①→②→③の順序で実行されます．

この他，クロックパルスをトリガにして，ある処理を順次実行したい場合などにも process 文を使用することができます．

図 3.21 process 文の例

第3章

process 文は，アーキテクチャ宣言の機能宣言部に記述します．もちろん，複数の process 文を記述することも可能です．たとえば，図 3.22 に示すようなコードを考えてみましょう．信号 *S1* と *S2* が同時に変化したとすると，代入文 1 と代入文 2，および二つの process 文は，同時に並列処理されます．その際，各 process 文の中の代入文 3 と代入文 4，また代入文 5 と代入文 6 はそれぞれ順次実行されることになります．

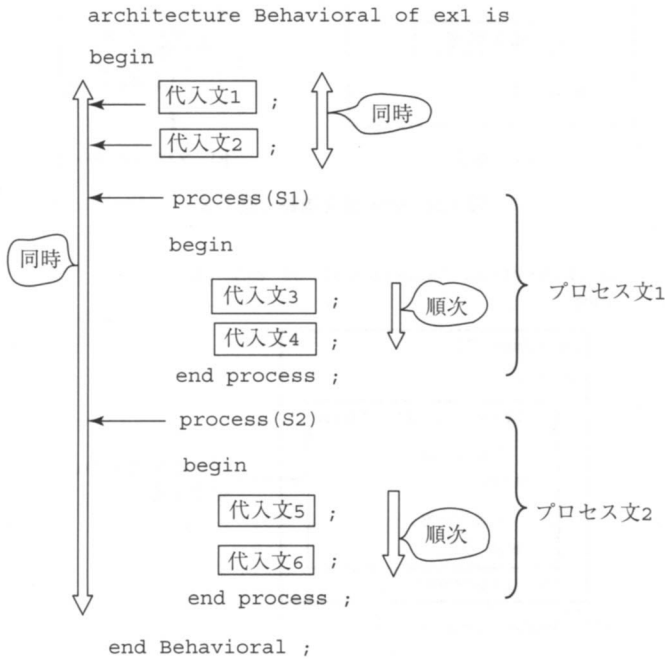



図 3.22 実行順序の例

▶3.2.2 if 文

if（イフ）文は，ある条件が成立するか否かで実行する処理を選択する構文です．選択された処理は，シーケンシャルに実行されます．図 3.23 に，if 文の書式とフローチャートを示します．if 文の考え方は，C などのプログラミング言語と同様です．

(a) 書式　(b) フローチャート

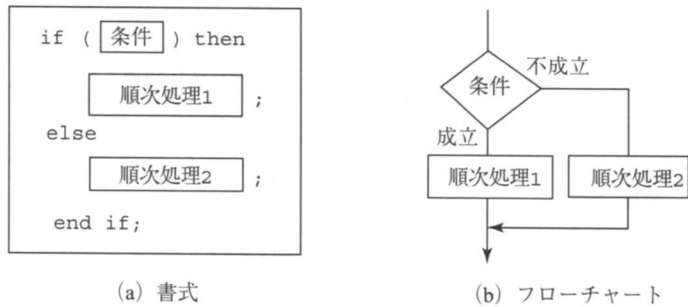



図 3.23　if 文

図 3.23 において，else 部は，必要なければ省略することが可能です．else 部を省略すると，ある条件が成立した場合のみ順次処理を実行し，条件が不成立の場合には何もせずに if 文を終了します（図 3.24 参照）．

(a) 書式　(b) フローチャート

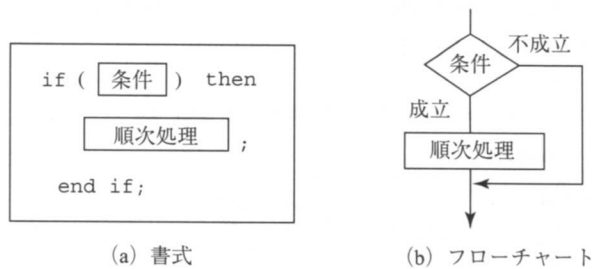



図 3.24　else 部を省略した if 文

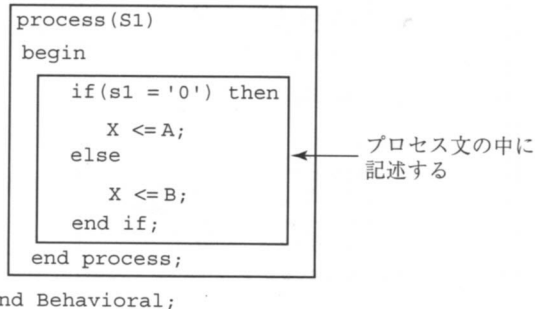



図 3.25　if 文の記述場所

if 文は，シーケンシャルに実行される処理ですから，アーキテクチャ宣言の機能宣言部（コンカレント領域）に直接記述せずに，process 文の中に記述します（図 3.25 参照）．このことは，後で学ぶ case 文も同じです．

複数の条件を順番に判定することで，実行する順次処理を選択したい場合には，elsif（エルスイフ）文を使用します．図 3.26 に，elsif 文の書式とフローチャートを示します．elsif 文では，elsif のスペルに注意しましょう（elseif ではない）．

(a) 書式　(b) フローチャート

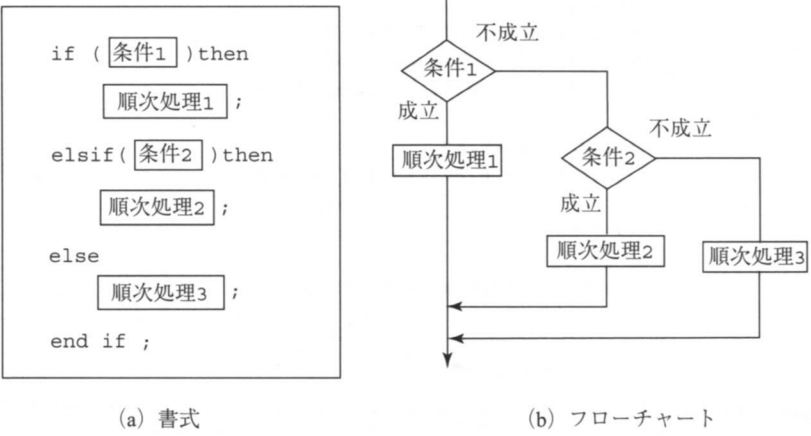



図3.26 elsif文

if文やelsif文の条件は，表3.5に示す関係演算子を用いて記述します．

表3.5 関係演算子

演算子	機能
=	等しい
/=	等しくない
<	右辺より小さい
>	右辺より大きい
<=	右辺以下
>=	右辺以上

つぎに，process文とif文を使う例題をみてみましょう．図3.27に示すAND/ORセレクタ回路を考えます．選択信号Sが0のときは出力Fに入力AとBのAND，Sが0のときは出力Fに入力AとBのORが出力される回路です．VHDLのコードでは，選択信号Sの値をif文で判定してANDかORの動作を選択します．また，process文のセンシティビティリストには，入力A，Bおよび，選択信号Sを記述します．リスト3.13に，if文を用いたコードの記述例を示します．

A	B	S	F
0	0	0	0
0	1	0	0
1	0	0	0
1	1	0	1
0	0	1	0
0	1	1	1
1	0	1	1
1	1	1	1

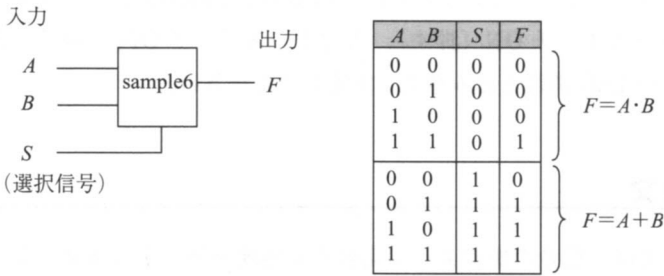



図3.27 AND/ORセレクタ回路

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity sample6 is
    Port ( A,B,S : in std_logic;
                F : out std_logic);
end sample6;

architecture Behavioral of sample6 is
begin

process(A,B,S)
begin
    if(S = '0') then
            F <= A and B ;
    else
            F <= A or B ;
    end if;
end process;

end Behavioral;
```


リスト 3.13　if 文を使った AND/OR セレクタ回路

表 3.6　ピン割り当て（HDL トレーナー）

記号		ピン番号	備考
入力	*A*	35	SW2
	B	36	SW3
	S	34	SW1
出力	*F*	61	LED4 の g

process 文のセンシティビティリストには，すべての入力信号 *A,B,S* を記述することに注意してください．

HDL トレーナーを使用して，表 3.6 のようなピン割り当てを行い実習してみましょう．HDL トレーナーのプッシュスイッチと 7 セグメント LED は，負論理動作をしますので，図 3.27 の真理値表を，入力 0 →スイッチ ON（押す），入力 1 →スイッチ OFF（離す）および，出力 0 → LED 点灯，出力 1 → LED 消灯と読み替えて実習してください．

▶3.2.3　case 文

case（ケイス）文は，信号の値によって実行する処理を選択する構文です．図 3.28 に，case 文の書式とフローチャートを示します．

(a) 書式　(b) フローチャート

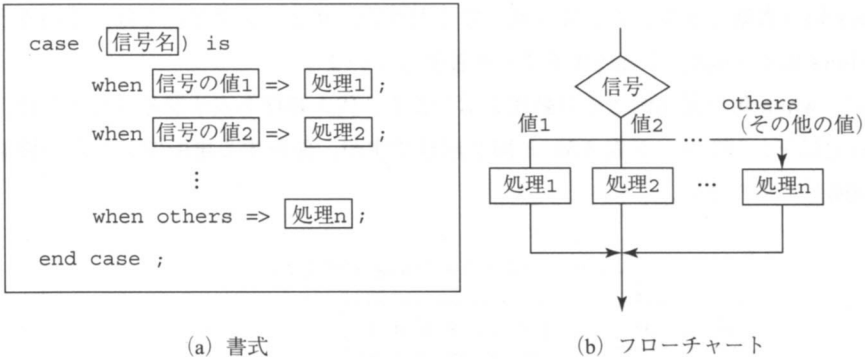



図 3.28　case 文

信号が,「信号の値1」ならば処理1,「信号の値2」ならば処理2を実行します.信号の値が,記述したどれにも該当しない場合には,when others以下の処理nを実行してcase文を終了します.

case文もif文と同様process文の中に記述します.しかし,if文では処理がシーケンシャルに実行されましたが,case文ではすべてのwhen以下の判定がコンカレントに並列実行されます.したがって,when以下に記述する「信号の値」には,同じ値があってはいけません.また,when others句を用いて,式がとりうるすべての値を記述することが必要です.

リスト3.14は,図3.27に示したAND/ORセレクタ回路をcase文で記述したコードです.このコードでは,選択信号のとりうる値(0と1)をすべて記述してありますが,この場合でも

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity sample7 is
   Port ( A,B,S : in std_logic;
          F : out std_logic);
end sample7;

architecture Behavioral of sample7 is
begin

process(A,B,S)
begin
case (S) is
        when '0' => F <= A and B ;
        when '1' => F <= A or B ;
        when others => null ;
end case;
end process;

end Behavioral;
```

センシティビティリスト

case文はprocess文の中に書く

S = '0' の場合の代入処理

S = '1' の場合の代入処理

when others句は省略できない

リスト 3.14　case 文を使った AND/OR セレクタ回路

when others 句は省略できないことに注意してください．ザイリンクス社では，このような場合の when others 句に「null」と記述することを推奨しています．

図 3.29 に，when 句の記述例とその動作を示します．代入処理を表す記号「<=」は，関係演算子の「右辺以下」（73 ページ表 3.5）と同じ記号ですが，使用する場所でどちらの機能を示しているか判断してください．

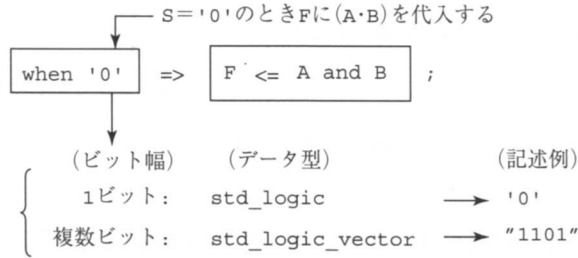



図 3.29　when 句の記述例と動作

また，リスト 3.13 やリスト 3.14 の式（選択信号 S）のデータ型は「std_logic」で指定していたので信号の値にはシングルコーテーション「'」を使用しました．もし，データ型として複数ビットをまとめて扱う「std_logic_vector」を指定した式の場合には，ダブルコーテーション「"」を使用して信号の値を記述します．

▶3.2.4　動作記述と構造記述

図 3.30 は，前に扱った AND/OR セレクタ回路（図 3.27）と同じものです．

この回路は，真理値表から求めた論理式をカルノー図などによって簡単化することで，図 3.31 のようなディジタル回路で構成できることがわかります．

図 3.30 と図 3.31 は同じ動作をする回路ですが，図 3.30 ではブロック図と真理値表を用いて動作を表しているのに対して，図 3.31 では具体的な回路の構造を表しています．VHDL でコードを記述する際にも，同様に二種類の書き方が考えられます．図 3.30 のように回路の動作を考えた書き方を**動作記述**，図 3.31 のように回路の構造を考えた書き方を**構造記述**といいます．リス

A	B	S	F
0	0	0	0
0	1	0	0
1	0	0	0
1	1	0	1
0	0	1	0
0	1	1	1
1	0	1	1
1	1	1	1

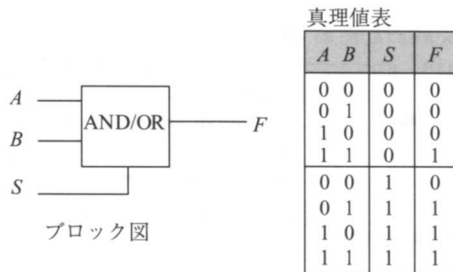



図 3.30　AND/OR セレクタ回路（動作図）

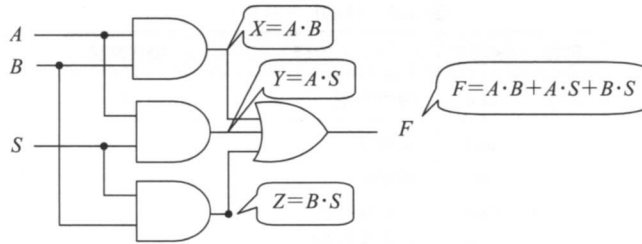



図3.31 AND/ORセレクタ回路（構造図）

ト3.15にcase文を使用した動作記述（リスト3.14と同じ），リスト3.16にsignal文を使用した構造記述によるコードを示します．動作記述を行えることは，VHDLを用いた設計の大きな長所の一つです．

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity sample7 is
    Port ( A,B,S : in std_logic;
                F : out std_logic);
end sample7;

architecture Behavioral of sample7 is
begin

process(A,B,S)
begin
   case (S) is
          when '0' => F <= A and B ;
          when '1' => F <= A or B ;
          when others => null ;
   end case;
end process;

end Behavioral;
```

リスト3.15 動作記述

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity sample8 is
    Port ( A,B,S : in std_logic;
                F : out std_logic);
end sample8;

architecture Behavioral of sample8 is

signal X,Y,Z : std_logic;

begin

X <= A and B ;
         Y <= A and S ;
         Z <= B and S ;
         F <= X or Y or Z ;

end Behavioral;
```

リスト3.16 構造記述

▶3.2.5 VHDLの演算子と予約語

表3.7に，VHDLで使用できる演算子とその優先順位の一覧を示します．優先順位の数字は，1が最も優先順位が高く，数字が大きくなっていくほど優先順位が低くなってきます．優先順位の変更には，カッコ「(　)」を使用します．

表3.8に，VHDLの予約語一覧を示します．ここに示した予約語は，入力や出力の信号名などに使用することはできません．

表 3.7　VHDL の演算子

種類	演算子	機能	優先順位
論理	not	論理否定	1
	and or nand nor xor xnor	論理積 論理和 否定論理積 否定論理和 排他的論理和 否定排他的論理和	6
算術	abs **	絶対値 べき乗	1
	＋ －	正の符号 負の記号	3
	＋ －	加算 減算	4
	* / mod rem	乗算 除算 被除数と同符号の余り 除数と同符号の余り	2
関係	＝ /＝ < > <＝ >＝	等しい 等しくない 右辺より小さい 右辺より大きい 右辺以下 右辺以上	5
連接	&	ビットの結合	4

＊優先順位は 1 が最も高い

表 3.8　VHDL の予約語

abs access after alias all and architecture array assert attribute begin block body buffer bus case component con guration constant disconnect downto else elsif end entity exit file for function generate generic group guarded if impure in inertial inout is label library linkage literal loop map mod nand new next nor not null of on open or others out package port postponed procedural procedure process portected pure range record reference register reject rem report return rol ror select severity signal shared sla sll sra srl subtype then to transport type unaffected units until use variable wait when while with xnor xor

3.3 各種の組合せ回路

これまで，組合せ回路を記述するために必要な基礎知識について学んできました．ここでは，加算器や減算器，デコーダやエンコーダなど簡単な各種の組合せ回路を設計してみましょう．例題形式で問題を示しますので，まずは自分で考えてみてください．

また，できるだけ実際の開発ツールや評価ボードを用いた実習を行いながら学習を進めてください．ただし，HDLトレーナーのプッシュスイッチや7セグメントLEDなどは，負論理動作をしますので，各真理値表の，入力0→スイッチON（押す），入力1→スイッチOFF（離す）および，出力0→LED点灯，出力1→LED消灯などと読み替えて実習を行ってください．

3.3.1 加算器と減算器

半加算器（HA : half adder）は，1ビットどうしの加算を行う回路です．図3.32に半加算器の真理値表と回路を示します．

真理値表

A	*B*	*S*	*C*
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

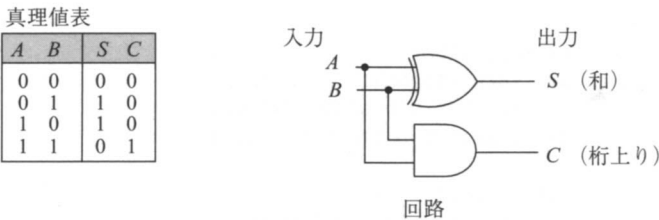



図3.32　半加算器

例題3−1

半加算器をVHDLで記述しなさい．ただし，構造記述とすること．

解答例

論理演算子を使用して記述したコードをリスト3.17に示します．表3.9に示すピン割り当てを行い実習で動作を確認してみましょう．

表3.9　例題3−1のピン割り当て

記号		ピン番号	備考
入力	*A*	34	SW1
	B	35	SW2
出力	*S*	74	LED3のg
	C	61	LED4のg

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity rei3_1 is
    Port ( A : in std_logic;
           B : in std_logic;
           S : out std_logic;
           C : out std_logic);
end rei3_1;

architecture Behavioral of rei3_1 is

begin

          S <= A xor B ;
          C <= A and B ;

end Behavioral;
```

リスト 3.17　半加算器のコード（構造記述）

例題 3−2

半加算器を VHDL で記述しなさい．ただし，算術演算子「＋」を用いた動作記述とすること．

解答例

論理演算子を使用して記述したコードをリスト 3.18 に示します．

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
-- use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity rei3_2 is
  Port ( A : in std_logic;
         B : in std_logic;
         S : out std_logic;
         C : out std_logic);
end rei3_2;

architecture Behavioral of rei3_2 is
         signal ADD : std_logic_vector(1 downto 0);
begin
         ADD <= ('0' & A) + ('0' & B);
         S <= ADD(0);
         C <= ADD(1);

end Behavioral;
```

リスト 3.18　半加算器のコード（動作記述）

入力 $A + B$ を算術的に加算した結果は，2 ビットになります．その下位ビットが和 S，上位ビットが桁上がり C です．リスト 3.18 ①では，2 ビットの内部信号 ADD を宣言しています．VHDL では，代入される側とする側のビット幅は同じでなければなりません．したがって，ADD と同じビット幅になるように，連接演算子&を用いて，

入力 A と B を2ビットに変換してから加算しています（リスト3.18②）．その後，ADDの下位ビットとADD（0）上位ビットADD（1）をそれぞれ出力 S と C に代入しています（リスト3.18③）．

これまでは，ライブラリ宣言でつぎの三つのパッケージを宣言していました．

STD_LOGIC_1164.ALL ・・・・・データ型を使用するためなどに必要

STD_LOGIC_ARITH.ALL ・・・・・符号なし，符号あり演算の両方をするために必要

STD_LOGIC_UNSIGNED.ALL ・・・・符号なし演算をするために必要

ここでは，符号なし演算に限定した処理を行いたいので，パッケージSTD_LOGIC_ARITH.ALLは，コメント文にしておきます（リスト3.18④）．

表3.9に示すピン割り当てを行い実習で動作を確認してみましょう．また，「View RTL Schematic」によって論理合成した回路を表示すると，図3.33のようになっており，図3.32と等価であることが確認できるはずです．

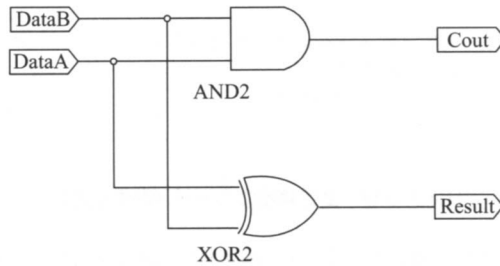



図3.33 リスト3.18の論理合成結果（RTL Schematic）

半加算器は，上位ビットに桁上がり信号 C を渡すことはできますが，下位ビットからの桁上がり信号を受け取ることはできません．つまり，複数ビットどうしの演算には，使用することができない半人前の加算器です．これに対して，全加算器（FA : full adder）は，下位ビットからの桁上がり信号 C_i を受け取ることのできる一人前の加算回路です．図3.34に，全加算器の真理値表とブロック図を示します．

真理値表

A	B	C_i	S	C_o
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

入力 出力

A B C_i FA S C_o

ブロック図

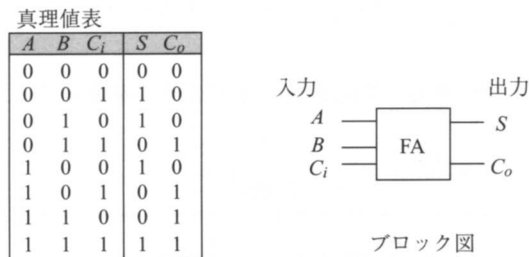



図3.34 全加算器

例題3-3

全加算器をVHDLで記述しなさい．ただし，算術演算子「+」を用いた動作記述とすること．

解答例

リスト3.19に全加算器のコードを示します．

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
--use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity rei3_3 is
   Port ( A : in std_logic;
          B : in std_logic;
          C_I : in std_logic;
          S : out std_logic;
          C_O : out std_logic);
end rei3_3;

architecture Behavioral of rei3_3 is
          signal ADD : std_logic_vector(1 downto 0) ;
begin
          ADD <= ('0' & A) + ('0' & B) + ( '0' & C_I) ;
          S <= ADD(0) ;
          C_O <= ADD(1) ;

end Behavioral;
```

リスト 3.19　全加算器のコード（動作記述）

半加算器と比べると，入力信号 C_i が増えていますが，基本的な記述は同じです．表 3.10 に示すピン割り当てを行い実習で動作を確認してみましょう．

表 3.10　例題 3-3 のピン割り当て

記号		ピン番号	備考
入力	A	34	SW1
	B	35	SW2
	C_6	36	SW3
出力	S	74	LED3 の g
	C_o	61	LED4 の g

全加算器は，半加算器 2 個を使用して図 3.35 に示すように構成することもできます．

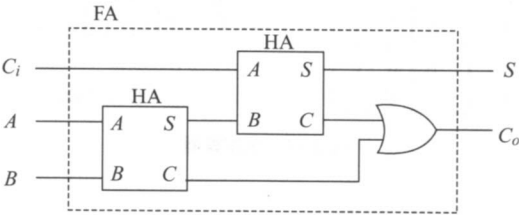



図 3.35　半加算器を用いた全加算器の構成

例題 3-4

半加算器 2 個を用いた全加算器を VHDL で記述しなさい．ただし，構造記述とすること．

解答例

この全加算器の回路は，図3.36に示すようになります．

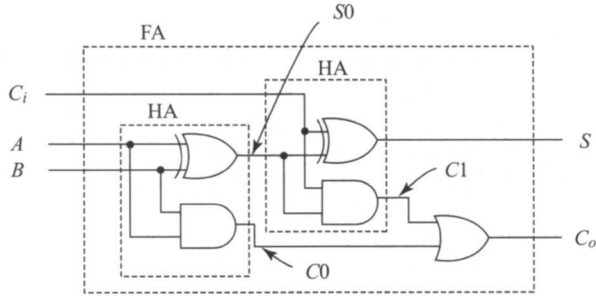



図3.36 半加算器を用いた全加算器の回路

前段の半加算器の出力を内部信号 *S0* と *C0*，後段の半加算器の出力を内部信号 *C1* として記述したコードをリスト3.20に示します．表3.10に示すピン割り当てを行い実習で動作を確認してみましょう．

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity rei3_4 is
    Port ( A : in std_logic;
           B : in std_logic;
           C_I : in std_logic;
           S : out std_logic;
           C_O : out std_logic);
end rei3_4;

architecture Behavioral of rei3_4 is
        signal S0,C0,C1: std_logic;
begin
        S0 <= A xor B ;
        C0 <= A and B ;
        C1 <= C_I and S0 ;
        S <= C_I xor S0 ;
        C_O <= C1 or C0 ;
end Behavioral;
```

リスト3.20 全加算器のコード（構造記述）

半減算器（HS : half subtracter）は，1ビットどうしの減算を行う回路です．図3.37に半減算器の真理値表と回路を示します．

真理値表

A	B	D	B_o
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

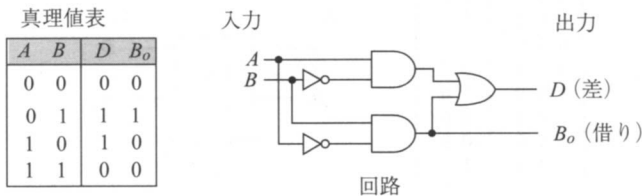



図3.37 半減算器

例題 3-5

半減算器を VHDL で記述しなさい．ただし，構造記述とすること．

解答例

リスト 3.21 に半減算器のコードを示します．表 3.10 に示すピン割り当てを行い実習で動作を確認してみましょう．

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity rei3_5 is
    Port ( A : in std_logic;
           B : in std_logic;
           D : out std_logic;
           B_O : out std_logic);
end rei3_5;

architecture Behavioral of rei3_5 is
           signal X, Y:std_logic ;
begin
           X <= A and (not B) ;
           Y <= B and (not A) ;
           D <= X or Y ;
           B_O <= Y ;
end Behavioral;
```

リスト 3.21　半減算器のコード（構造記述）

表 3.11　例題 3-5 のピン割り当て

記号		ピン番号	備考
入力	A	34	SW1
	B	35	SW2
出力	D	74	LED3 の g
	B_o	61	LED4 の g

半減算器は，上位ビットに桁借り信号 B を渡すことはできますが，下位ビットからの桁借り信号を受け取ることはできません．これに対して，全減算器（FS : full subtracter）は，下位ビットからの桁借り信号 B_i を受け取ることのできる減算回路です．図 3.38 に，全減算器の真理値表とブロック図を示します．

真理値表

$A\ B\ B_i$	$D\ B_o$
0 0 0	0 0
0 0 1	1 1
0 1 0	1 1
0 1 1	0 1
1 0 0	1 0
1 0 1	0 0
1 1 0	0 0
1 1 1	1 1

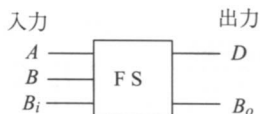



図 3.38　全減算器

例題 3-6

全減算器を VHDL で記述しなさい．ただし，算術演算子「−」を用いた動作記述とすること．

解答例

リスト 3.22 に全減算器のコードを示します．2 ビットの内部信号 *SUB* を宣言しています（リスト 3.22 ①）．*SUB* と同じビット幅になるように，連接演算子&を用いて，入力 *A*，*B*，*B_I* を 2 ビットに変換してから減算しています（リスト 3.22 ②）．その後，*SUB* の下位ビット *SUB*（0）と上位ビット *SUB*（1）をそれぞれ出力 *D* と *B_O* に代入しています（リスト 3.22 ③）．また，パッケージ IEEE.STD_LOGIC_ARITH.ALL は，コメント文を用いて無効にしておきましょう（リスト 3.22 ④）．

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
--use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity rei3_6 is
    Port ( A : in std_logic;
           B : in std_logic;
           B_I : in std_logic;
           D : out std_logic;
           B_O : out std_logic);
end rei3_6;

architecture Behavioral of rei3_6 is
           signal SUB : std_logic_vector(1 downto 0);
begin
           SUB <= ('0' & A) - ('0' & B) - ( '0' & B_I);
           D <= SUB(0);
           B_O <= SUB(1);
end Behavioral;
```

リスト 3.22　全減算器のコード（動作記述）

表 3.12 に示すピン割り当てを行い実習で動作を確認してみましょう．

表 3.12　例題 3-6 のピン割り当て

記号		ピン番号	備考
入力	A	34	SW1
	B	35	SW2
	B_i	36	SW3
出力	D	74	LED3 の g
	B_o	61	LED4 の g

▶3.3.2　エンコーダとデコーダ

エンコーダ（encoder）は，人が見て意味のわかるデータをディジタル回路用のデータに変換する働きをする回路で，符号器ともよばれます．たとえば，10 進数のデータを 2 進数に変換するエンコーダなどがあります．図 3.39 に，10 進-2 進エンコーダの例を示します．このエンコー

ダは，10 進数の 0 ～ 3 までに対応させた 4 本の入力ピン A_3，A_2，A_1，A_0 のうち，信号 1 を入力したピンを有効とします．そして，入力した 10 進数を 2 進数に変換して，出力ピン F_1，F_0 から 2 ビットで出力します．

真理値表

A_3	A_2	A_1	A_0	F_1	F_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

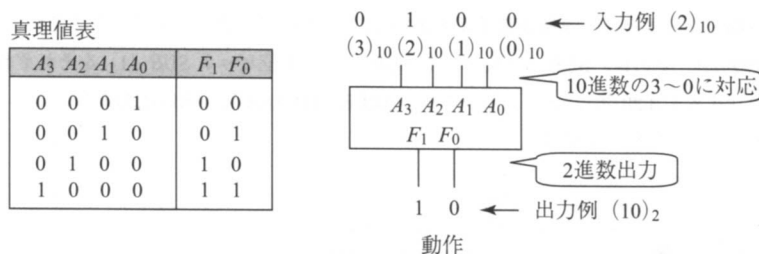



図 3.39　10 進-2 進エンコーダの例

例題 3-7

図 3.39 に示したエンコーダを VHDL で記述しなさい．ただし，動作記述とすること．

解答例

リスト 3.23 にエンコーダのコードを示します．入力と出力のデータ型には，ベクトル型 std_logic_vector を使用しました．そして，case 文を用いて入力データを判定し，出力データ型を決めています（74 ページ参照）．この例題では，4 ビット入力（16 通りのデータ）のうちの 4 通りのデータのみを使用していますので，未使用入力については when others 句で F に "11" を代入しました．また，case 文は process 文の中に記述することに注意してください．process 文のセンシティビティリストには，入力信号 A を記述します．

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity rei3_7 is
  Port ( A : in std_logic_vector(3 downto 0);
         F : out std_logic_vector(1 downto 0));
end rei3_7;

architecture Behavioral of rei3_7 is
begin
         process(A)
         begin
               case A is
                       when "0001" => F <= "00" ;
                       when "0010" => F <= "01" ;
                       when "0100" => F <= "10" ;
                       when "1000" => F <= "11" ;
                       when others => F <= "11" ;
               end case;
         end process;
end Behavioral;
```

リスト 3.23　エンコーダのコード（動作記述）

表 3.13 に示すピン割り当てを行い実習で動作を確認してみましょう．

表 3.13　例題 3-7 のピン割り当て

記号		ピン番号	備考
入力	$A(0)$	37	SW4
	$A(1)$	36	SW3
	$A(2)$	35	SW2
	$A(3)$	34	SW1
出力	$F(0)$	61	LED4 の g
	$F(1)$	74	LED3 の g

デコーダ（decoder）は，ディジタル回路用のデータを人が見て意味のわかるデータに変換する働きをする回路で，復号器ともよばれます．つまり，エンコーダとは逆の働きをする回路であると考えることができます．たとえば，2 進数のデータを 10 進数に変換するデコーダなどがあります．図 3.40 に，2 進-10 進デコーダの例を示します．このデコーダは 2 ビットの 2 進数入力 A_1，A_0 を 10 進数に変換します．変換結果は，10 進数の 3 ～ 0 までに対応させた 4 本の出力ピン F_3，F_2，F_1，F_0 から出力します．

真理値表

A_1	A_0	F_3	F_2	F_1	F_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

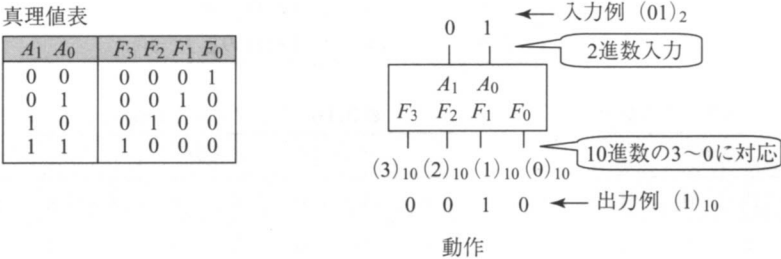



図 3.40　2 進-10 進デコーダの例

例題 3-8

図 3.40 に示したデコーダを VHDL で記述しなさい．ただし，動作記述とすること．

解答例

リスト 3.24 にデコーダのコードを示します．動作の流れとしては，リスト 3.23 で学んだエンコーダとほとんど同じです．

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity rei3_8 is
  Port ( A : in std_logic_vector(1 downto 0);
         F : out std_logic_vector(3 downto 0));
end rei3_8;

architecture Behavioral of rei3_8 is
begin
```

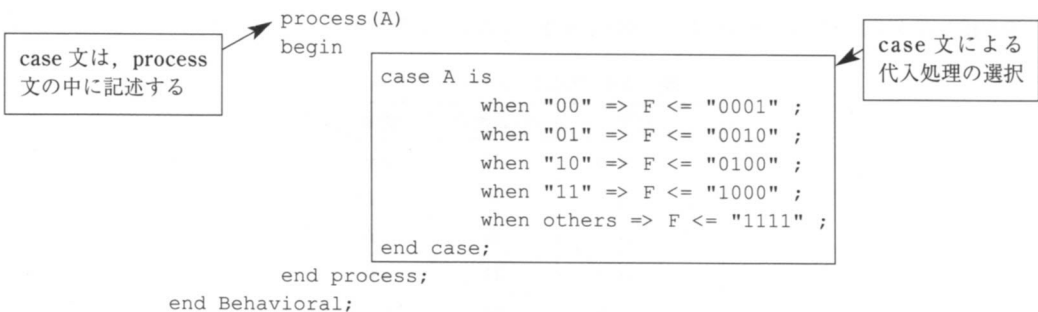
入力 A，出力 F をベクトル型で宣言

リスト 3.24　デコーダのコード（動作記述）

表 3.14 に示すピン割り当てを行い実習で動作を確認してみましょう．

表 3.14　例題 3-8 のピン割り当て

記号		ピン番号	備考
入力	*A*(1)	34	SW1
	A(0)	35	SW2
出力	*F*(3)	97	LED1 の g
	F(2)	85	LED2 の g
	F(1)	74	LED3 の g
	F(0)	61	LED4 の g

表 3.15　ロータリスイッチの動作

位置	出力			
	H_3	H_2	H_1	H_0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
A	1	0	1	0
B	1	0	1	1
C	1	1	0	0
D	1	1	0	1
E	1	1	1	0
F	1	1	1	1

表 3.16　7 セグメント LED の表示（0 で点灯）

入力								表示
a	b	c	d	e	f	g	dp	
0	0	0	0	0	0	1	1	0
1	0	0	1	1	1	1	1	1
0	0	1	0	0	1	0	1	2
0	0	0	0	1	1	0	1	3
1	0	0	1	1	0	0	1	4
0	1	0	0	1	0	0	1	5
0	1	0	0	0	0	0	1	6
0	0	0	1	1	0	1	1	7
0	0	0	0	0	0	0	1	8
0	0	0	0	1	0	0	1	9
0	0	0	1	0	0	0	1	A
1	1	0	0	0	0	0	1	b
0	1	1	0	0	0	1	1	C
1	0	0	0	0	1	0	1	d
0	1	1	0	0	0	0	1	E
0	1	1	1	0	0	0	1	F

つぎに，デコーダの応用回路として7セグメントLEDに数字を表示させてみましょう．HDLトレーナーには，16進数のロータリスイッチが搭載されています．このスイッチは，表3.15に示すような動作をします．一方，図3.41に7セグメントLEDのセグメント記号，表3.16に表示データを示します．

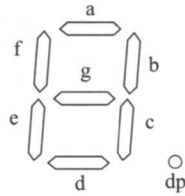



図3.41 セグメント記号

例題3-9

表3.15と表3.16を参考にして，ロータリスイッチから入力した16進数のデータ通りの数字を7セグメントLEDで表示する回路をVHDLで記述しなさい．ただし，動作記述とすること．

解答例

リスト3.25に7セグメントLED用デコーダのコードを示します．記述のパターンは，リスト3.24のデコーダと同様です．ロータリスイッチからの入力信号にはベクトル型の*RSW*（0）～*RSW*（3），7セグメントLEDへの出力信号には*LED*（0）～*LED*（7）を宣言しました．case文では，最後にwhen others句の記述を忘れないようにしましょう．リスト3.25では，when others句に「null」と記述しましたが，たとえば，

「when others => LED <= "11111111" ;」

などのように記述しても結構です．

この例題のような回路を設計する場合，従来のディジタル回路設計法では，表3.15や表3.16などから回路の動作を論理式で表し，その後に論理式の簡単化などを行って回路の構造を設計する必要がありました．しかし，VHDLによる設計では，ただちに動作記述を行うことができます．

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity rei3_9 is
  Port ( RSW : in std_logic_vector(3 downto 0);
         LED : out std_logic_vector(7 downto 0));
end rei3_9;

architecture Behavioral of rei3_9 is
begin
         process(RSW)
         begin
                case RSW is
                       when "0000" => LED <= "00000011" ;
                       when "0001" => LED <= "10011111" ;
                       when "0010" => LED <= "00100101" ;
                       when "0011" => LED <= "00001101" ;
                       when "0100" => LED <= "10011001" ;
                       when "0101" => LED <= "01001001" ;
                       when "0110" => LED <= "01000001" ;
```

ロータリスイッチからの4ビット入力

7セグメントLEDへの8ビット出力

```
                    when "0111" => LED <= "00011011" ;
                    when "1000" => LED <= "00000001" ;
                    when "1001" => LED <= "00001001" ;
                    when "1010" => LED <= "00010001" ;
                    when "1011" => LED <= "11000001" ;
                    when "1100" => LED <= "01100011" ;
                    when "1101" => LED <= "10000101" ;
                    when "1110" => LED <= "01100001" ;
                    when "1111" => LED <= "01110001" ;
                    when others => null ;
                end case;
        end process;
end Behavioral;
```

when others 句を記述する

リスト3.25　7セグメントLED用デコーダのコード（動作記述）

表3.17に示すピン割り当てを行い実習で動作を確認してみましょう．

表3.17　例題3-9のピン割り当て

	記号	ピン番号	備考	
入力	*RSW*(0)	33	HEX SW0	
	RSW(1)	32	HEX SW1	
	RSW(2)	30	HEX SW2	
	RSW(3)	29	HEX SW3	
出力	*LED*(0)	50	LED 4 (LED D)	dp
	LED(1)	61		g
	LED(2)	60		f
	LED(3)	58		e
	LED(4)	56		d
	LED(5)	55		c
	LED(6)	53		b
	LED(7)	52		a

3.3.3　マルチプレクサとデマルチプレクサ

マルチプレクサ（multiplexer）は，セレクタ（selector）ともよばれ，複数のデータの中から一つのデータを選択する回路です．入力データ数 $2^n = m$ の中から n ビットの選択信号線を使用して一つの出力を選択する回路を $m \times 1$ マルチプレクサといいます．図3.42に，4×1マルチプレクサの例を示します．

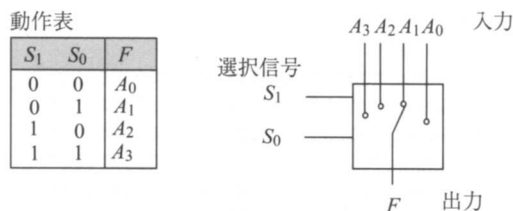



図3.42　4×1マルチプレクサ

例題 3－10

4 × 1 マルチプレクサを VHDL で記述しなさい．ただし，動作記述とすること．

解答例

リスト 3.26 に，4 × 1 マルチプレクサのコードを示します．

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity rei3_10 is
  Port ( A : in std_logic_vector(3 downto 0);
         S : in std_logic_vector(1 downto 0);
         F : out std_logic);
end rei3_10;

architecture Behavioral of rei3_10 is
begin
         process(A,S)
         begin
                 case S is
                         when "00" => F <= A(0) ;
                         when "01" => F <= A(1) ;
                         when "10" => F <= A(2) ;
                         when "11" => F <= A(3) ;
                         when others => null ;
                 end case;
         end process;
end Behavioral;
```

入力データ 4 ビット

選択信号 2 ビット

出力データ 1 ビット

リスト 3.26　4 × 1 マルチプレクサのコード（動作記述）

表 3.18 に示すピン割り当てを行い実習で動作を確認してみましょう．表 3.18 では，4 ビットの入力信号 *A*（3）～ *A*（0）に，HDL トレーナーのロータリスイッチを割り当てています．たとえば，表 3.19 に示すように，ロータリスイッチを「9」の位置に設定した場合の入力 *A*（3）～ *A*（0）は「1001」となり，2 ビットの選択信号に割り当てた SW1 と SW2 の操作によって，*A*（3）～ *A*（0）のいずれかが選択されて *F* へ出力されます．ここでも，HDL トレーナーのプッシュスイッチ SW1 と SW2 は押した場合に信号「0」，7 セグメント LED は信号「0」で点灯することに注意しましょう．

表 3.18　例題 3－10 のピン割り当て

記号		ピン番号	備考
入力	*A*(3)	33	HEX SW3
	A(2)	32	HEX SW2
	A(1)	30	HEX SW1
	A(0)	29	HEX SW0
	S(1)	34	SW1
	S(0)	35	SW2
出力	*F*	61	LED4 の g

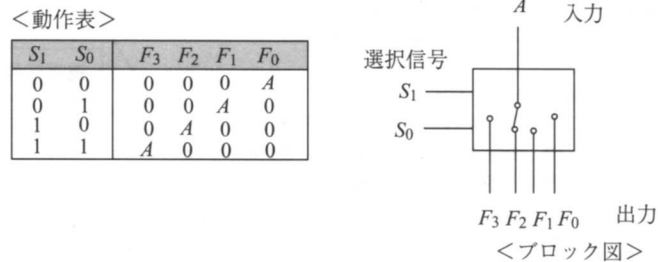
表 3.19　4 × 1 マルチプレクサの動作例

位置	*A*(3)	*A*(2)	*A*(1)	*A*(0)	*S*(1)	*S*(0)	*F*	
9	1	0	0	1	0	0	1	*A*(0)
					0	1	0	*A*(1)
					1	0	0	*A*(2)
					1	1	1	*A*(3)

デマルチプレクサ（demultiplexer）は，1ビットの入力データを複数ビットの出力線の中のいずれかに出力する回路です．つまり，マルチプレクサとは逆の働きをする回路だと考えることができます．n ビットの選択信号線を使用して，1ビットの入力を $2^n = m$ ビットの出力線へ出力する回路を $1 \times m$ デマルチプレクサといいます．図3.43に，1×4 デマルチプレクサの例を示します．

S_1	S_0	F_3	F_2	F_1	F_0
0	0	0	0	0	A
0	1	0	0	A	0
1	0	0	A	0	0
1	1	A	0	0	0

図3.43　1 × 4 デマルチプレクサ

第3章

例題 3-11

1 × 4 デマルチプレクサをVHDLで記述しなさい．ただし，動作記述とすること．

解答例

リスト3.27に，1 × 4 デマルチプレクサのコードを示します．case文を用いてwhen句で出力 F に代入するデータを選択しています．出力 F は，4ビットのベクトル型で宣言していますので，出力 F に入力 A を代入する場合には連接演算子を用いて同じ4ビット幅にしていることに注意してください．

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity rei3_11 is
  Port ( A : in std_logic;
         S : in std_logic_vector(1 downto 0);
         F : out std_logic_vector(3 downto 0));
end rei3_11;

architecture Behavioral of rei3_11 is
begin
        process(A,S)
        begin
                case S is
                        when "00" => F <= "000" & A ;
                        when "01" => F <= "00" & A & '0' ;
                        when "10" => F <= '0' & A & "00" ;
                        when "11" => F <= A & "000" ;
                        when others => null ;
                end case ;
        end process ;
end Behavioral;
```

入力データ1ビット

選択信号2ビット

出力データ4ビット

4ビットのデータとして代入

リスト3.27　1 × 4 デマルチプレクサのコード（動作記述）

たとえば，リスト 3.28 のような，出力 F の選択された 1 ビットのみに入力 A を代入するコードを考えてみましょう．このコードでは，図 3.44 に示すように，代入の行われなかった出力 F の上位 3 ビットの値が不定（0 か 1 かが定まらない状態）となってしまいます．したがって，論理合成を行うと「WARNING」メッセージが出ます．誤動作の原因となる可能性が高いので，このような記述は避けましょう．

when "00" => F(0) <= A;

F(3)	F(2)	F(1)	F(0)
?	?	?	↑ A

（F(3)～F(1)：不定）

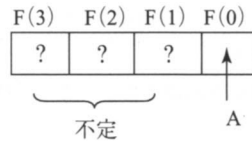



図 3.44 出力 F への代入

悪いコードの例

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity rei3_11 is
  Port ( A : in std_logic;
         S : in std_logic_vector(1 downto 0);
         F : out std_logic_vector(3 downto 0));
end rei3_11;

architecture Behavioral of rei3_11 is
begin
        process(A,S)
        begin
                case S is
                        when "00" => F(0) <= A ;
                        when "01" => F(1) <= A ;
                        when "10" => F(2) <= A ;
                        when "11" => F(3) <= A ;
                        when others => null ;
                end case ;
        end process ;
end Behavioral;
```

代入されなかったビットの値は「不定」となる

リスト 3.28 出力 F の 1 ビットのみに入力 A を代入するコード

リスト 3.28 のコードを用いて，表 3.20 に示すピン割り当てを行い実習で動作を確認してみましょう．

表 3.20 例題 3-11 のピン割り当て

記号		ピン番号	備考
入力	A	37	SW4
	$S(1)$	34	SW1
	$S(0)$	35	SW2
出力	$F(3)$	97	LED1 の g
	$F(2)$	85	LED2 の g
	$F(1)$	74	LED3 の g
	$F(0)$	61	LED4 の g

▶ 演習問題 3

3.1　つぎのデータ型の違いについて説明しなさい.

①「std_logic」

②「std_logic_vector」

3.2　シーケンシャル文とコンカレント文の違いについて説明しなさい.

3.3　process 文のセンシティビティリストについて説明しなさい.

3.4　構造記述と動作記述について説明しなさい.

3.5　図 3.45 は, 3 ビットデータの一致を調べる回路である. この回路を VHDL で構造記述したリスト 3.29 の①～⑧に適当な語句を答えなさい.

A	B	C	F
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

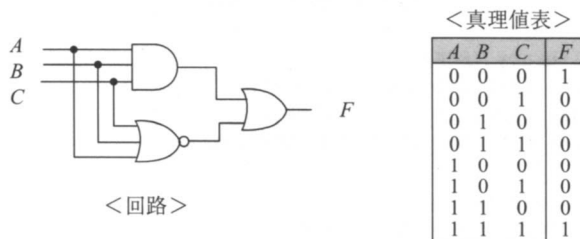



図 3.45　3 ビット一致回路

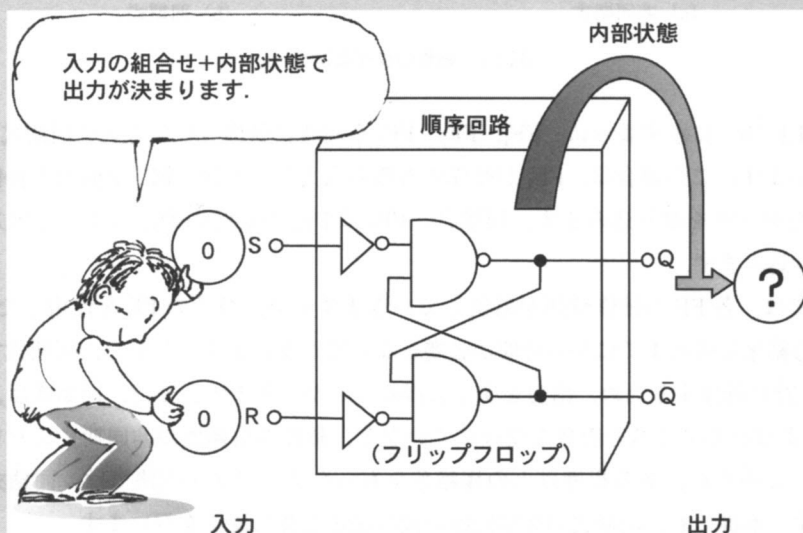
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity syo3_5 is
    Port ( A : ① ;
           B : ② ;
           C : ③ ;
           F : ④ ) ;
end syo3_5;

architecture Behavioral of syo3_5 is
           ⑤ X,Y : std_logic ;
⑥
           X <= A and B and C ;
           Y <= ⑦(A or B or C) ;
           F <= ⑧ ;
end Behavioral;
```

リスト 3.29　3 ビット一致回路のコード（構造記述）

3.6　図 3.45 に示した 3 ビット一致回路の VHDL コードを動作記述しなさい. ただし, 入力データはベクトル型を使用すること.

第4章 順序回路の設計

順序回路は，入力の組合せに加えて，回路の内部状態がどのようになっているかで出力が決まります．順序回路の中心は，フリップフロップとよばれるラッチ回路です．フリップフロップ（flip-flop）とは，「パタンパタン」という音や，ビーチサンダルのことを表す英語です．この意味のように，フリップフロップは，二つの安定状態をもち，何かの信号をきっかけにして，一方の安定状態から他方の安定状態に遷移する回路です．

この章では，非同期式と同期式の違いや各種のフリップフロップの動作について確認した後，シフトレジスタやカウンタなどの順序回路を設計しましょう．順序回路の設計では，主に同期式回路の動作記述が用いられます．

4.1 フリップフロップの設計

順序回路は，内部にラッチ（latch）回路（記憶回路ともいいます）を含みます．ラッチ回路としては，フリップフロップを使用します．この節の初めでは，順序回路が動作するタイミングやクロックのタイミングなどについて理解しましょう．その後，各種のフリップフロップをVHDLコードで記述する方法などについて学びましょう．

▶4.1.1 非同期式と同期式

複数のフリップフロップ（以下FFと省略表記します）を使用して構成した回路の動作を考えましょう．たとえば，3個のFFを直列に接続（カスケード接続）して動作させる場合，図4.1（a）に示すように，$FF_1 \to FF_2 \to FF_3$と順次動作させる方式を非同期式（asynchronous）といいます．この場合は，FF_1が動作してデータを出力してからFF_2がそのデータを受け取って動作し，その後同様にFF_3が動作します．つまり，ドミノ倒しのようなイメージで動作します．

（a）非同期式　　（b）同期式

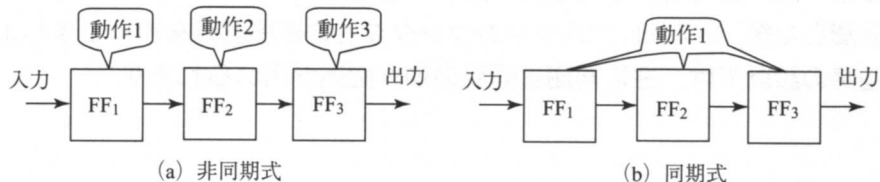



図4.1　動作のタイミング

一方，図4.1（b）に示すように，FF_1，FF_2，FF_3を同時に動作させる方式を同期式（synchronous）といいます．この場合は，FF_2が動作する際の入力データは，FF_1が前回の動作によって出力していたデータを取り込みます．同様に，FF_3はFF_2が前回の動作によって出力していたデータを取り込みます．

非同期式では，各FFの動作時間が累積していきますので，多くのFFを使用した場合には，最終的な出力結果を得るまでに長い時間が必要となってしまいます．しかし，同期式では，すべてのFFが一斉に動作するため，出力をすぐに得ることができます．また，同期式は，ハザード（hazard）とよばれるノイズの影響を受けにくいなど，動作の信頼性が非同期式よりも高いことが利点です．このため，ある程度以上の複雑さをもったディジタル回路では，同期式が主流となっています．本書でも，同期式の順序回路の設計法を主体に説明を行います．

▶4.1.2　クロックの記述

クロック（clock）は，回路が動作するタイミングのきっかけになる信号です．通常は，図4.2に示すような方形波の立ち上りエッジ（ポジティブエッジ）または，立ち下りエッジ（ネガティブエッジ）を使用します．つまり，たとえばポジティブエッジを指定した場合には，方形波が0から1に立ち上がる瞬間ごとに，回路が1回動作します．

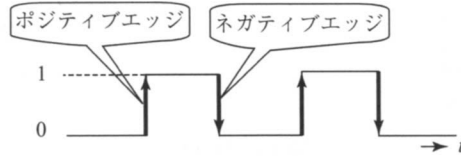



図4.2　クロックの例

図4.1に示した非同期式と同期式の回路では，たとえば，クロック入力の仕方が図4.3のようになります．同期式では，有効な1個のクロックですべてのFFが一斉に動作します．

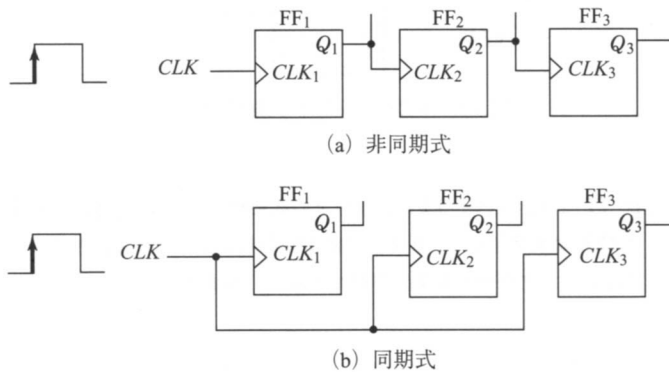
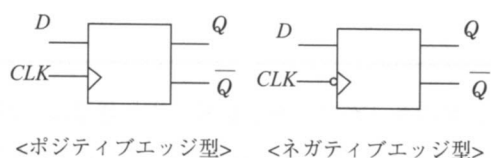



図4.3　クロック入力の例（ポジティブエッジ）

VHDLでは，クロック *CLK* のポジティブエッジとネガティブエッジをつぎのように記述します．通常は，この記述をprocess文と組合せて使用しますが，具体的な使用法については後で学びます．

CLK' event and CLK = '1'……ポジティブエッジ
CLK' event and CLK = '0'……ネガティブエッジ

▶4.1.3　*D*–FF

図4.4に，*D*–FFの図記号などを示します．*D*–FFは，有効なクロックが入力されたときに，入力 *D* のデータを取り込んで出力 *Q* から出力します．*D*–FFの *D* は，ディレイ（delay：遅れる）という意味に由来します．クロックのポジティブエッジとネガティブエッジの図記号についても確認しておいてください．

(a) 図記号

D	Q	$\overline{Q}$
0	0	1
1	1	0

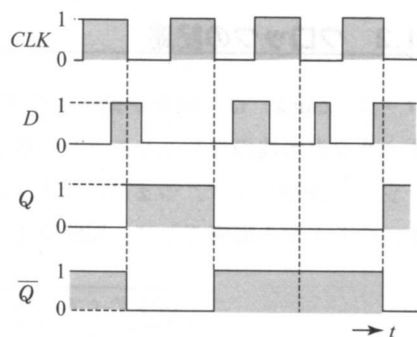
(b) 真理値表

(c) タイムチャート例
(ネガティブエッジ型)

図4.4　D-FF

例題 4-1

図4.5に示すD-FFをVHDLで記述しなさい．

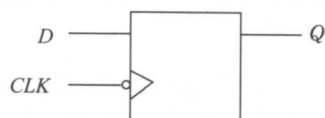



図4.5　ネガティブエッジ型D-FF

解答例

ネガティブエッジ型のD-FFです．リスト4.1に，コードの記述例を示します．process文のセンシティビティリストに記述したクロック*CLK*が変化すると，if文の判定が行われます．そこで，ネガティブエッジのときにif文の条件式が成立し，入力Dのデータがそのまま出力Qから出力されます．

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity rei4_1 is
    Port ( D : in std_logic;
           CLK : in std_logic;
           Q : out std_logic);
end rei4_1;

architecture Behavioral of rei4_1 is
begin
       process(CLK)
              begin
                     if (CLK'event and CLK='0') then
                            Q <= D ;
                     end if ;
       end process ;
end Behavioral;
```

*CLK*のネガティブエッジでif文の条件が成立する

リスト4.1　D-FFのコード

論理合成を行い，「View RTL Schematic」を実行すると，図 4.6 に示す回路が構成されたことが確認できます．

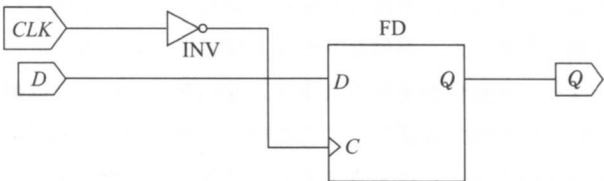



図 4.6　構成された回路

HDL トレーナーを用いて，表 4.1 に示すピン割り当てによって実習を行います．初めに，プッシュスイッチ SW1 で入力 *D* の値を決めた後，プッシュスイッチ SW2 を操作してクロック *CLK* を入力し，7 セグメント LED4 の点灯によって出力 *Q* の値を確認しましょう．

表 4.1　例題 4-1 のピン割り当て

記号		ピン番号	備考
入力	*D*	34	SW1
	CLK	35	SW2
出力	*Q*	61	LED4 の g

FF には，リセット端子やセット端子の付いたタイプがあります．図 4.7 に，リセット端子付きの *D*-FF の図記号を示します．この *D*-FF は，クロック *CLK* のポジティブエッジで動作し，リセット *RESET* 端子（負論理）に 0 が入力されるとリセット信号が有効になります．

ただし，リセット動作には，非同期リセットと同期リセットがあることに注意してください．非同期リセットは，有効なリセット信号がリセット端子に入力されると，すぐに出力 *Q* がリ

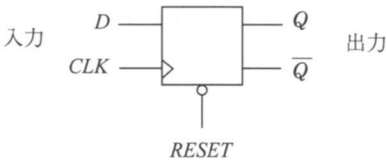



図 4.7　リセット端子付き *D*-FF

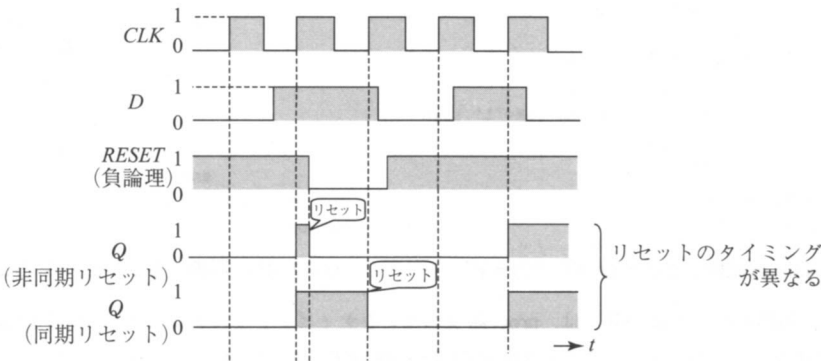



図 4.8　非同期リセットと同期リセット

セット（0が出力）されます．したがって，非同期リセットのことを強制リセットとよぶこともあります．一方，同期リセットでは，有効なリセット信号をリセット端子に入力しても，それだけではリセット動作は起こりません．有効なリセット信号の入力に加えて，有効なクロック信号が入力された場合に，リセット動作が実行されます．非同期リセットと同期リセットの動作の違いを，図4.8のタイムチャート例で確認してください．また，図4.7の図記号からは，非同期リセットと同期リセットを区別することはできませんので注意が必要です．

例題 4-2

図4.9に示すD-FFをVHDLで記述しなさい．ただし，リセット方式が非同期と同期それぞれのコードを記述しなさい．

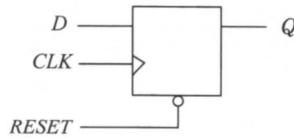



図4.9　リセット端子付きD-FF

解答例

非同期リセット端子付きFFのコードをリスト4.2，同期リセット端子付きFFのコードをリスト4.3に示します．

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity rei4_2_1 is
    Port ( D : in std_logic;
          CLK : in std_logic;
          RESET : in std_logic;
          Q : out std_logic);
end rei4_2_1;

architecture Behavioral of rei4_2_1 is
begin
          process(CLK,RESET)
          begin
                  if RESET = '0' then
                          Q <= '0' ;
                  elsif CLK'event and CLK='1' then
                          Q <= D ;
                  end if ;
          end process ;
end Behavioral;
```

リスト4.2　非同期リセットD-FFのコード

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity rei4_2_2 is
    Port ( D : in std_logic;
           CLK : in std_logic;
           RESET : in std_logic;
           Q : out std_logic);
end rei4_2_2;

architecture Behavioral of rei4_2_2 is
begin
           process(CLK)
           begin
                   if CLK'event and CLK='1' then
                           if RESET = '0' then
                                   Q <= '0' ;
                           else
                                   Q <= D ;
                           end if ;
                   end if ;
           end process ;
end Behavioral;
```

リスト4.3　同期リセットD-FFのコード

リスト4.3の同期リセットD-FFでは，process文のセンシティビティリストにリセット信号***RESET***を記述する必要がありません．また，if文での判定順序が***CLK→RESET***となっていることを確認してください．非同期リセット，同期リセットとも，表4.2に示すピン割り当てによって実習を行い，動作の違いを比較しましょう．

表 4.2　例題 4-2 のピン割り当て

記号		ピン番号	備考
入力	*D*	34	SW1
	CLK	35	SW2
	RESET	36	SW3
出力	*Q*	61	LED4 の g

FF には，イネーブル端子の付いたタイプがあります．イネーブル（enable）とは，動作を可能にするという意味をもちます．イネーブル端子に有効な信号を入れているときにはクロック入力を受け付けますが，それ以外では出力 *Q* を保持したまま動作しません．リスト **4.4** に，正論理のイネーブル端子付き *D*-FF（クロックはネガティブエッジ型）のコード（process 文）を示します．


```
process(CLK)
begin
        if (CLK'event and CLK = '0') then
                if (ENABLE = '1') then
                        Q <= D ;
                end if ;
        end if ;
end process ;
```

リスト 4.4　イネーブル端子付き *D*-FF のコード（process 文）

例題 4-3

図 **4.10** に示すイネーブル端子付きの *D*-FF を VHDL で記述しなさい．ただし，同期リセット型とする．

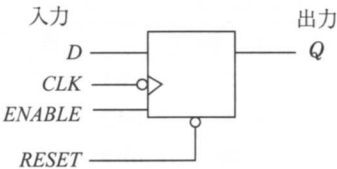



図 4.10　イネーブル端子付き *D*-FF

解答例

同期リセット端子（負論理）とイネーブル端子（正論理）をもったネガティブエッジ型のクロックで動作する *D*-FF に関する例題です．この *D*-FF のコードをリスト 4.5，フローチャートを図 **4.11** に示します．

これまで使用してきた，*CLK*，*RESET*，*ENABLE* などの記号は，VHDL の予約語ではありませんので，他の記号に変更することも可能です．

表 4.3 に示すピン割り当てを行って実習で動作を確認しましょう．イネーブル端子への入力はロータリスイッチの最下位ビット（HEXSW0）を割り当てました．ロータリスイッチの表示が 0 でイネーブル無効，1 で有効となります．

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity rei4_3 is
   Port ( D : in std_logic;
          CLK : in std_logic;
          ENABLE : in std_logic;
          RESET : in std_logic;
          Q : out std_logic);
end rei4_3;

architecture Behavioral of rei4_3 is
begin
        process(CLK)
        begin
                if (CLK'event and CLK = '0') then
                         if (ENABLE = '1') then
                                 if (RESET = '0') then
                                         Q <= '0' ;
                                 else
                                         Q <= D ;
                                 end if ;
                         end if ;
                end if ;
        end process ;
end Behavioral;
```

リスト 4.5　イネーブル端子付き *D*-FF のコード

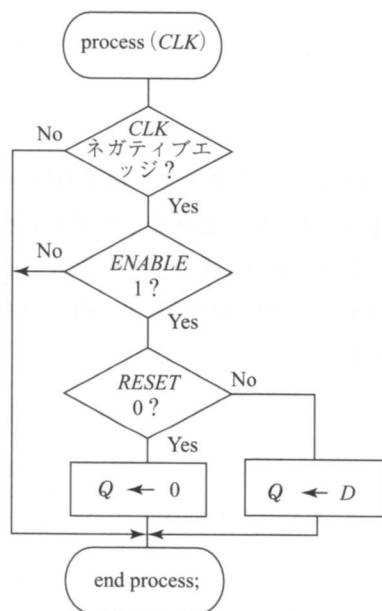



図 4.11　イネーブル端子付き *D*-FF のフローチャート

表 4.3　例題 4-3 のピン割り当て

記　号		ピン番号	備　考
入力	*D*	34	SW1
	CLK	35	SW2
	RESET	36	SW3
	ENABLE	33	ロータリ SW の HEXSW ①
出力	*Q*	61	LED4 の g

▶ 4.1.4　*RS*-FF

RS-FF は，セット端子 *S* とリセット端子 *R* をもった FF です．図 4.12 に，ポジティブエッジ型のクロック端子付き *RS*-FF の図記号などを示します．

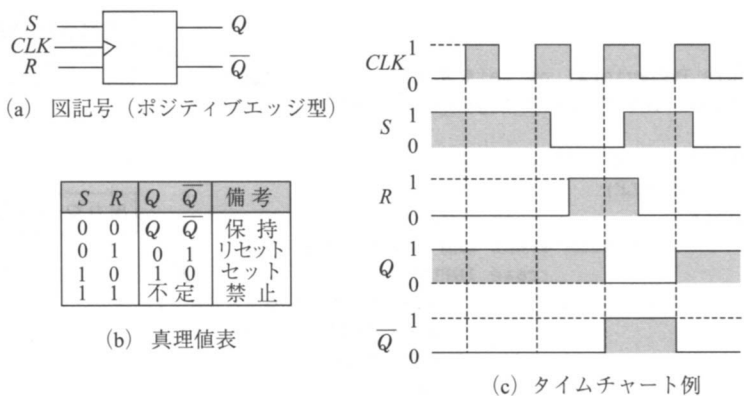



図 4.12 クロック端子付き *RS*-FF

例題 4-4

図 4.13 に示すクロック端子付き *RS*-FF を VHDL で記述しなさい.

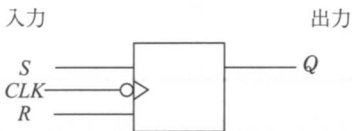



図 4.13 ネガティブエッジ型 *RS*-FF

解答例

クロック *CLK* のネガティブエッジで動作する *RS*-FF です. リスト 4.6 に, コードを示します. 入力端子 *S* と *R* を, signal 文で宣言した 2 ビットベクトル型の内部信号 *INPUT* に代入しています. そして, クロック *CLK* の立ち下り時に case 文を用いて出力 *Q* への代入処理を行っています. 表 4.4 示すピン割り当てを行って実習で動作を確認しましょう.

表 4.4 例題 4-4 のピン割り当て

記号		ピン番号	備考
入力	*S*	34	SW1
	R	35	SW2
	CLK	36	SW3
出力	*Q*	61	LED4 の g

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity rei4_4 is
    Port ( S : in std_logic;
           R : in std_logic;
           CLK : in std_logic;
           Q : out std_logic);
```

```
end rei4_4;

architecture Behavioral of rei4_4 is
        signal INPUT : std_logic_vector(1 downto 0) ;
begin
        INPUT <= (S & R) ;
        process(CLK)
        begin
                if (CLK'event and CLK = '0') then
                        case INPUT is
                                when "01" => Q <= '0' ;
                                when "10" => Q <= '1' ;
                                when others => null ;
                        end case ;
                end if ;
        end process ;
end Behavioral;
```

内部信号 *INPUT* をベクトル型で宣言

出力 Q への代理処理

リスト 4.6　クロック端子付き *RS*-FF のコード

▶4.1.5　*JK*-FF

RS-FF は，$R = S = 1$ の入力を行うと出力 Q が不定になるため，この入力を禁止しています．*JK*-FF は，この点を改良した FF であり，$J = K = 1$ の入力を行うと出力 Q は反転動作をします．*JK*-FF の端子 J と K は，それぞれ *RS*-FF の端子 S と R に対応します．図 4.14 に，ネガティブエッジ型のクロック端子付き *JK*-FF の図記号などを示します．

J	K	Q	$\overline{Q}$	備　考
0	0	Q	$\overline{Q}$	保　持
0	1	0	1	リセット
1	0	1	0	セット
1	1	$\overline{Q}$	Q	反　転

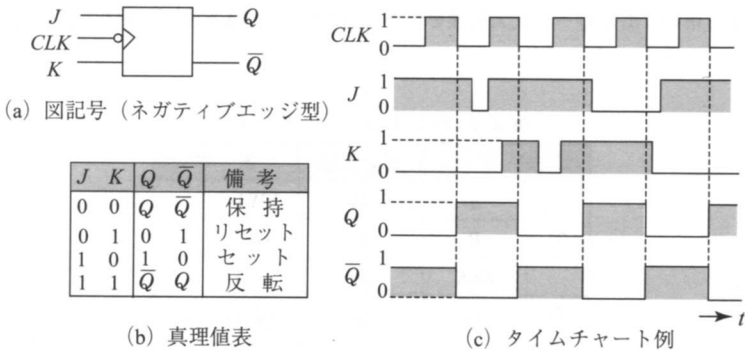



図 4.14　クロック付き *JK*-FF

例題 4-5

図 4.15 に示すクロック端子付き *JK*-FF を VHDL で記述しなさい．

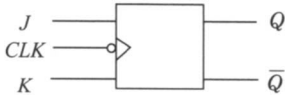



図 4.15　クロック端子付き *JK*-FF

解答例

クロック *CLK* のネガティブエッジで動作する *JK*-FF です．出力端子は，*Q* とその反転 $\overline{Q}$ の 2 本です．リスト 4.7 に，この *JK*-FF のコードを示します．基本的な記述法は，リスト 4.6 の *RS*-FF のコードと同じです．入力端子 *J* と *K* を，signal 文で宣言した 2 ビットベクトル型の内部信号 *INPUT* に代入しています．そして，クロック *CLK* の立ち下り時に case 文を用いて内部信号 *WORK* への代入処理を行っています．

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity rei4_5 is
   Port ( J : in std_logic;
          K : in std_logic;
          CLK : in std_logic;
          Q : out std_logic;
          Q_NOT : out std_logic);
end rei4_5;

architecture Behavioral of rei4_5 is
        signal INPUT : std_logic_vector(1 downto 0) ;
        signal WORK : std_logic   ;
begin
        INPUT <= (J & K) ;
        process(CLK)
        begin
        if (CLK'event and CLK = '0') then
                 case INPUT is
                         when "01" => WORK <= '0' ;
                         when "10" => WORK <= '1' ;
                         when "11" => WORK <= not WORK ;
                         when others => null ;
                 end case ;
        end if ;
   end process ;
   Q <= WORK ;
   Q_NOT <= not WORK ;
end Behavioral;
```

内部信号 *INPUT* の宣言

内部信号 *WORK* の宣言

内部信号 *WORK* への代入処理

出力 *Q* と *Q_NOT* への代入処理

リスト 4.7 クロック端子付き *JK*-FF のコード

出力信号は，直接的に not などの操作を記述することはできません．つまり，リスト 4.7 の出力への代入処理において，「Q_NOT <= not Q ;」とは記述できないのです．したがって，内部信号 *WORK* を宣言して，「Q_NOT <= not WORK ;」のように記述しています．表 4.5 に示すピン割り当てを行って実習で動作を確認しましょう．

表 4.5 例題 4-5 のピン割り当て

記 号		ピン番号	備 考
入力	*J*	34	SW1
	K	35	SW2
	CLK	36	SW3
出力	*Q*	74	LED3 の g
	Q_NOT	61	LED4 の g

この実習では，入力に$J=K=1$を設定してクロック信号*CLK*を有効にした際，つまりHDLトレーナーでは，プッシュスイッチSW1とSW2を離した状態でSW1を押した場合に，時として不安定な動作をすることがあります．不安定な動作とは，7セグメントLEDの点灯がうまく反転しないことを指します．これは，プッシュスイッチを操作する際に生じるノイズ（チャタリングといいます）の影響です．プッシュスイッチを1回しか押していないつもりでも，チャタリングが発生して複数回のクロックが入力されてしまうためです．チャタリングはディジタル回路の誤動作の原因となりますので，信頼性の高い回路を設計する場合には，チャタリング除去回路を設ける必要があります．

例題 4－6

図4.16に示すイネーブル端子付き*JK*－FFをVHDLで記述しなさい．

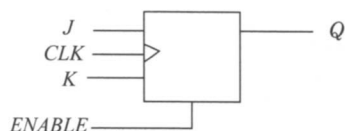



図4.16　イネーブル端子付き *JK*－FF

解答例

クロック*CLK*のポジティブエッジで動作する正論理のイネーブル端子付き*JK*－FFです．リスト4.8に，この*JK*－FFのコードを示します．リスト4.7と比較して理解するとよいでしょう．続いて，表4.6に示すピン割り当てを行って実習で動作を確認しましょう．イネーブル端子への入力はロータリスイッチの最下位ビット（HEXSW0）を割り当てました．ロータリスイッチの表示が0でイネーブル無効，1で有効となります．

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity rei4_6 is
  Port ( J : in std_logic;
         K : in std_logic;
         CLK : in std_logic;
         ENABLE : in std_logic;
         Q : out std_logic);
end rei4_6;

architecture Behavioral of rei4_6 is
        signal INPUT : std_logic_vector(1 downto 0) ;   ← 内部信号の宣言
        signal WORK : std_logic     ;
begin
        INPUT <= (J & K) ;
        process(CLK)
        begin
                if (CLK'event and CLK = '1') then
                        if (ENABLE = '1') then   ← イネーブル信号の判定
                                case INPUT is   ← JK-FFの動作
                                        when "01" => WORK <= '0' ;
                                        when "10" => WORK <= '1' ;
                                        when "11" => WORK <= not WORK ;
                                        when others => null ;
```

```
                    end case ;
                end if ;
            end if ;
        end process ;
        Q <= WORK ;
    end Behavioral;
```


リスト 4.8　イネーブル端子付き *JK*-FF のコード

表 4.6　例題 4-6 のピン割り当て

	記　号	ピン番号	備　考
入力	*J*	34	SW1
	K	35	SW2
	CLK	36	SW3
	ENABLE	33	ロータリ SW の HEXSW 0
出力	*Q*	61	LED4 の g

▶4.1.6　*T*-FF

T-FF は，有効な信号が入力されるたびに，出力 *Q* の値を反転する FF です．*T*-FF の *T* は，トグル（toggle : 切り替える）という意味に由来しています．図 4.17 に，*T*-FF の図記号などを示します．

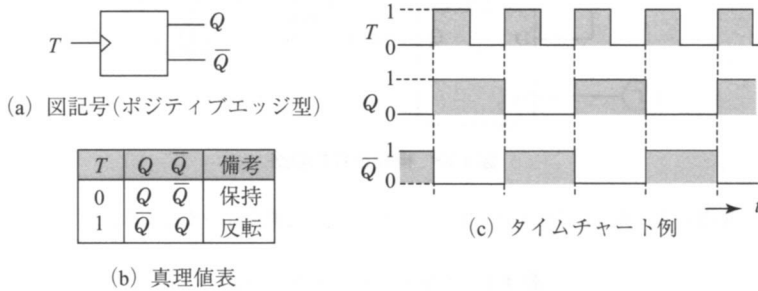



図 4.17　*T*-FF

例題 4-7

図 4.18 に示す *T*-FF を VHDL で記述しなさい．

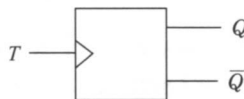



図 4.18　ポジティブエッジ型 *T*-FF

解答例

入力 *T* のポジティブエッジで動作する *T*-FF です．リスト 4.9 に，この *T*-FF のコードを示します．

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity rei4_7 is
  Port ( T : in std_logic;
         Q : out std_logic;
         Q_NOT : out std_logic);
end rei4_7;

architecture Behavioral of rei4_7 is
        signal WORK : std_logic ;
begin
        process(T)
        begin
                if (T'event and T = '1') then
                        WORK <= not WORK ;
                end if ;
        end process ;
        Q <= WORK ;
        Q_NOT <= not WORK ;
end Behavioral;
```

内部信号 *WORK* の宣言

WORK の反転処理

出力 *Q*，*Q_NOT* への代入処理

リスト 4.9　ポジティブエッジ型 *T*−FF のコード

論理合成を行い，「View RTL Schematic」を実行すると，図 4.19 に示すように，*D*−FF を用いた *T*−FF 回路が構成されたことが確認できます．

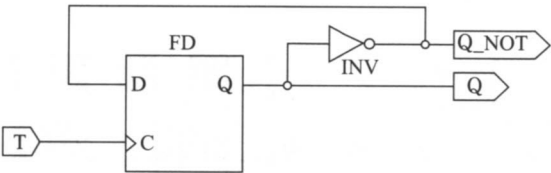



図 4.19　構成された回路

HDL トレーナーを用いて，表 4.7 に示すピン割り当てによって実習を行いましょう．

表 4.7　例題 4−7 のピン割り当て

記　号		ピン番号	備　考
入力	*T*	34	SW1
出力	*Q*	74	LED3 の g
	Q_NOT	61	LED4 の g

この実習では，例題 4−5 で説明したチャタリングが発生していることを意識しながら実習することも大切です．プッシュスイッチ操作時に発生するチャタリングの影響によって，正しい動作をしない場合があるはずです．確実なプッシュスイッチ操作を心がければチャタリングによる誤動作が減少することなどを体験してください．また，HDL トレーナーのプッシュスイッチは，押したときに信号が 1 → 0，離したときに信号が 0 → 1 に変化します．したがって，この実習で構成したポジティブエッジ型 *T*−FF では，プッシュスイッチを離したときに出力（7 セグメント LED の点灯）が変化することを確認しましょう．

4.2 シフトレジスタの設計

カスケード接続した複数のFFがクロック信号に同期して，データを1ビットずつ右方向または左方向に移動させる回路をシフトレジスタ（shift register）といいます．電光掲示板に表示された文字が，流れるように移動する回路もシフトレジスタを応用したものです．ここでは，VHDLによってシフトレジスタを構成する実習を行いましょう．

4.2.1 シリアルイン・パラレルアウトのシフトレジスタ

シフトレジスタは，入力と出力がシリアル（serial：直列）かパラレル（parallel：並列）かによって分類することができます．例として，図4.20にシリアルイン・パラレルアウトの4ビットシフトレジスタの構成とそのタイムチャート例を示します．

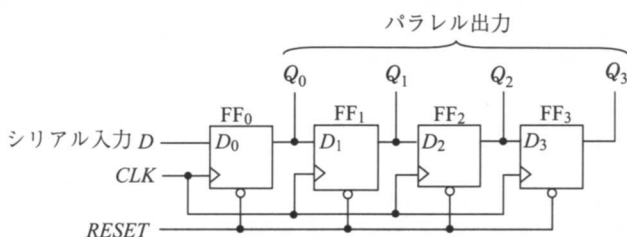
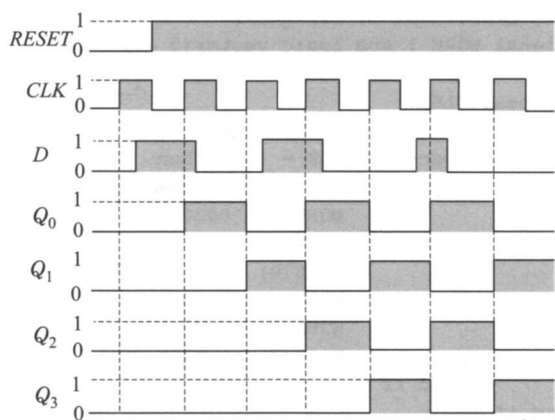



図4.20　シリアルイン・パラレルアウトのシフトレジスタ

このシフトレジスタは，クロック信号CLKのポジティブエッジ時ごとにデータを1ビットずつ右方向（$FF_0 \rightarrow FF_1 \rightarrow FF_2 \rightarrow FF_3$）へ移動していきます．使用している$D$-FFは，同期リセッ

ト端子付きでクロックのポジティブエッジで動作するタイプです．

例題 4−8

図 4.21 に示す 4 ビットのシリアルイン・パラレルアウトのシフトレジスタを VHDL で記述しなさい．ただし，D−FF は同期リセット型とすること．

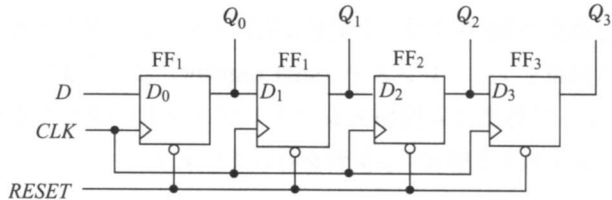



図 4.21　4 ビットのシリアルイン・パラレルアウトのシフトレジスタ

解答例

クロック入力 *CLK* のポジティブエッジで動作するシフトレジスタです．リスト 4.10 に，このシフトレジスタのコードを示します．

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity rei4_8 is
  Port ( D : in std_logic;
         CLK : in std_logic;
         RESET : in std_logic;
         Q : out std_logic_vector(3 downto 0));
end rei4_8;

architecture Behavioral of rei4_8 is
         signal WORK : std_logic_vector(3 downto 0) ;
begin
         process(CLK)
         begin
         if (CLK'event and CLK = '1') then
                  if (RESET = '0') then
                           WORK <= "0000" ;
                  else
                           WORK(0) <= D ;
                           WORK(1) <= WORK(0) ;
                           WORK(2) <= WORK(1) ;
                           WORK(3) <= WORK(2) ;
                  end if ;
         end if ;
         end process ;
         Q <= WORK ;
end Behavioral;
```


リスト 4.10　シリアルイン・パラレルアウトのシフトレジスタのコード

出力 Q と内部信号 *WORK* は，ベクトル型で宣言しました．この例題において，表 4.8 に示すようなピン割り当てを行い HDL トレーナーで実習を行うと，チャタリングの影響が強く現れるため動作の確認は困難でした．しかし，実際に実習してみることは，よい経験になることでしょう．

表 4.8 例題 4-8 のピン割り当て

記号		ピン番号	備考
入力	*D*	34	SW1
	CLK	35	SW2
	RESET	36	SW3
出力	$Q(0)$	97	LED1 の g
	$Q(1)$	85	LED2 の g
	$Q(2)$	74	LED3 の g
	$Q(3)$	61	LED4 の g

HDL トレーナーには，周波数 4 MHz のセラロック発振子が搭載されています．この発振子からの信号をクロックとして，シフトレジスタを動作させてみましょう．周波数 4 MHz の方形波は，図 4.22 に示すように，0.25 μs の周期 T で変化します（35 ページ，式（2.1）参照）．

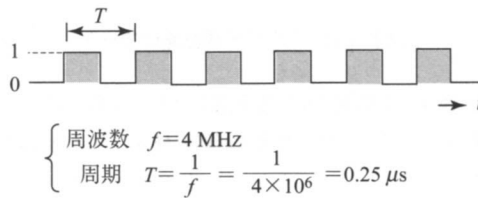



図 4.22 4 MHz の方形波

この周期でシフトレジスタを動作させたのでは，高速すぎて人の目では確認することができません．したがって，たとえば，セラロックからの方形波が 10 個入力されるたびに，1 個の方形波を出力するような回路を考えます（図 4.23）．このような回路を分周回路といいます． この例では，4 MHz の方形波は 10 分の 1 に分周されることになります．

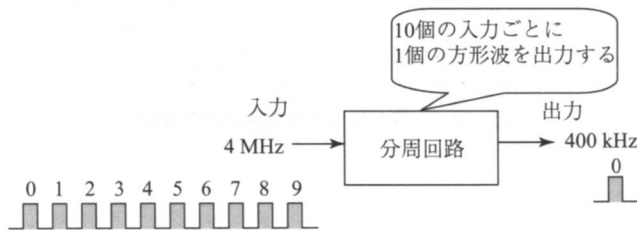



図 4.23 分周回路の考え方

周波数 4 MHz（4000 kHz）を 1/10 に分周すると 400 kHz になります．つまり，周期 0.25 μs は 10 倍の 2.5 μs になりますが，これでも人にとっては速すぎます．分周比をさらに大きくして，$1/2^{22}$ にすれば，次式のようにおよそ 1 s の周期 T' が得られます．

$$T' = (0.25 \times 10^{-6}) \times 2^{22} \fallingdotseq 1\,\mathrm{s}$$

この分周回路の VHDL コードの一部（process 文）をリスト 4.11 に示します．セラロックからのクロック信号を内部信号 *DD* によってカウントしています．そして，*DD* が $2^{22}-1$（1 を引いているのは，0 からカウントしているためです）になるたびに，方形波 *CE* を 1 個出力しています．

```
-- 分周回路       22 ビット
process (CLK)
begin
   if CLK'event and CLK='1' then
        DD <= DD + '1';
   end if;
end process;

process (DD)
begin
   if DD = "1111111111111111111111" then
      CE <= '1';
   else
      CE <= '0';
   end if;
end process;
```

クロック入力ごとに，*DD* に 1 を加算する

DD が $2^{22}-1$ になったら 1 個の方形波 *CE* を出力する

リスト 4.11　$1/2^{22}$ 分周回路のコード

リスト 4.11 の分周回路によって作成した周期 1 s の方形波をクロック信号にして，リスト 4.10 のシフトレジスタを動作させてみましょう．図 4.24 にシフトレジスタ実習回路の構成，リスト 4.12 に実習用コードを示します．

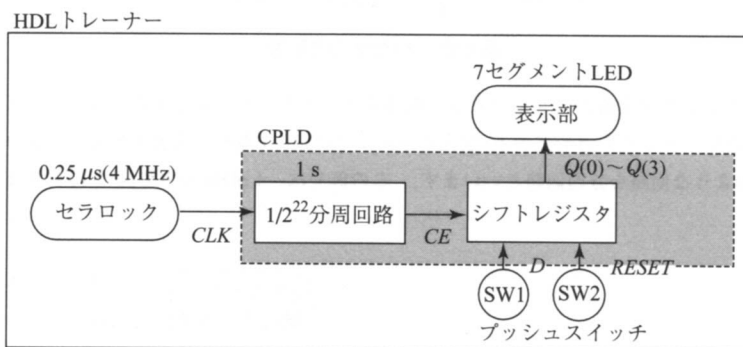



図 4.24　シフトレジスタ実習回路の構成

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity rei4_8 is
  Port ( D : in std_logic;
         CLK : in std_logic;
         RESET : in std_logic;
         Q : out std_logic_vector(3 downto 0));      内部信号の宣言
end rei4_8;

architecture Behavioral of rei4_8 is
        signal WORK : std_logic_vector(3 downto 0) ;
        signal DD : std_logic_vector(21 downto 0) ;
        signal CE : std_logic ;
begin

 -- 分周回路      22 ビット              分周回路のカウンタ部
process (CLK)
begin                                    クロックは CLK
  if CLK'event and CLK='1' then
      DD <= DD + '1';
  end if;
end process;
                                         分周回路の方形波発生部
process (DD)
begin
  if DD = "1111111111111111111111" then
     CE <= '1';
  else
     CE <= '0';
  end if;
end process ;

--  シフトレジスタ              シフトレジスタ部
process(CE)                                    クロックは CE
        begin
        if (CE'event and CE = '1') then        同期リセット
                 if (RESET = '0') then
                         WORK <= "0000" ;
                 else
                         WORK(0) <= D ;
                         WORK(1) <= WORK(0) ;
                         WORK(2) <= WORK(1) ;
                         WORK(3) <= WORK(2) ;
                 end if ;
        end if ;
        end process ;              出力 Q への代入処理
        Q <= WORK ;
end Behavioral;
```

リスト 4.12 シフトレジスタ（例題 4-8）の実習用コード

リスト 4.12 のコードを論理合成して「View Synthesis Report」を実行してみましょう．図 4.25 に，Synthesis Report の一部を示します．このコードで使用した FF は，シフトレジスタに 4 個，分周回路に 22 個，つまり合計 26 個であることが確認できます．

```
Cell Usage :
# BELS                    :50
#    AND2                 :24
#    AND3                 : 1
#                         : 1
#    AND8                 : 2
#    INV                  : 1
#    XOR2                 :21
# FlipFlops/Latches       :26
#    FD                   :26
# IO Buffers              : 7
#    IBUF                 : 3
#    OBUF                 : 4
```


図 4.25　Synthesis Report の一部

表 4.9 に示すピン割り当てによって実習を行いましょう．プッシュスイッチ SW1 を押し続ける（信号 $D = 0$）と 7 セグメント LED の表示が左から右へ順次点灯します．そして，プッシュスイッチ SW1 を離すと（信号 $D = 1$）と 7 セグメント LED の表示が左から右へ順次消灯します．また，プッシュスイッチ SW2 を押すと（$RESET = 0$），リセット信号が有効になり 4 個の 7 セグメント LED が点灯するはずです．この実習回路は，同期リセットなので SW2 は 1 秒（分周後のクロック *CE* の周期）以上押し続ける必要があります．

表 4.9　リスト 4.12 のピン割り当て

記　号		ピン番号	備　考
入力	*CLK*	22	CLK1
	D	34	SW1
	RESET	35	SW2
出力	*Q*(0)	97	LED1 の g
	Q(1)	85	LED2 の g
	Q(2)	74	LED3 の g
	Q(3)	61	LED4 の g

▶ 4.2.2　パラレルイン・シリアルアウトのシフトレジスタ

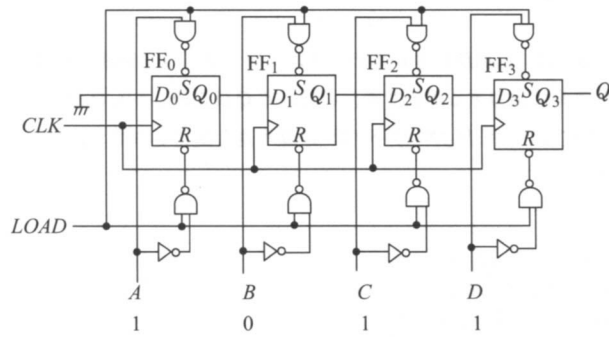
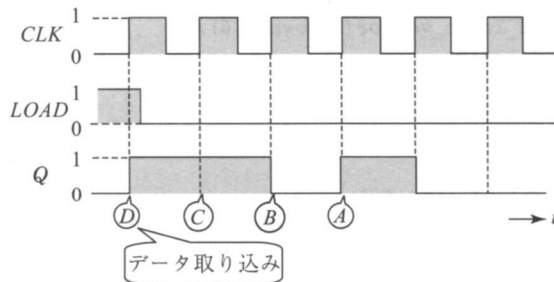
図 4.26 に，パラレルイン・シリアルアウトの 4 ビットシフトレジスタの構成とそのタイムチャート例を示します．使用している *D*-FF は，同期セット・リセット端子付きで，クロックがポジティブエッジで動作するタイプです．

(a) 回路例

(b) タイムチャート例

図 4.26 パラレルイン・シリアルアウトのシフトレジスタ

このシフトレジスタは，信号 *LOAD* が 1 のときクロック信号 *CLK* に同期して 4 ビットのパラレルデータ *ABCD* を各 FF に取り込みます．その後，クロック信号のポジティブエッジ時に各 FF のデータを右方向に 1 ビットずつシフトしていきます．つまり，出力 *Q* からは，入力されたパラレルデータが *D* → *C* → *B* → *A* の順でシリアルに出力されます．

例題 4－9

図 4.27 に示す 4 ビットのパラレルイン・シリアルアウトのシフトレジスタと同様の動作をする回路を VHDL で記述しなさい．ただし，使用する *D*－FF は，同期セット・リセット型とすること．

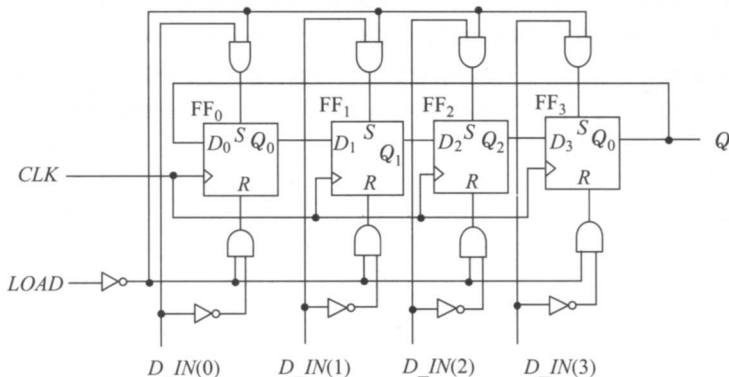



図 4.27 4 ビットのパラレルイン・シリアルアウトのシフトレジスタ

解答例

入力データ *D_IN*（0）～（3）は，*LOAD* 信号が0のときにクロック信号に同期して取り込まれます．FF_3 の出力 Q は，パラレル入力をシリアルに出力する端子です．この出力 Q は，FF_0 の入力 D_0 にフィードバック接続されていますので，出力 Q からは入力データ *D_IN*（0）～（3）が繰り返し出力されます．リスト 4.13 に，このシフトレジスタのコードを示します．

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity rei4_9 is
  Port ( D_IN : in std_logic_vector(3 downto 0);
         CLK : in std_logic;
         LOAD : in std_logic;
         Q : out std_logic );
end rei4_9;

architecture Behavioral of rei4_9 is
        signal WORK : std_logic_vector(3 downto 0) ;   -- 内部信号の宣言
begin
process(CLK)
        begin
        if (CLK'event and CLK = '1') then
                if (LOAD = '0') then                   -- CLKに同期して入力データを取り込む
                        WORK <= D_IN ;
                else
                        WORK(0) <= WORK(3) ;           -- シフトレジスタの処理
                        WORK(1) <= WORK(0) ;
                        WORK(2) <= WORK(1) ;
                        WORK(3) <= WORK(2) ;
                end if ;
        end if ;
        end process ;
        Q <= WORK(3) ;                                 -- シリアル出力Qへの代入処理
end Behavioral;
```

リスト 4.13　パラレルイン・シリアルアウトのシフトレジスタのコード

例題 4-8 と同様に，HDL トレーナーのセラロック 4 MHz を分周回路によって遅くして動作させる実習用コードをリスト 4.14，そのピン割り当てを表 4.10 に示します．パラレル入力データは，ロータリスイッチの出力 4 ビットを使用しています．ロータリスイッチで設定したデータが，およそ1秒ごとにシリアルに繰り返して出力される動作を確認してください．入力データの取り込みは，プッシュスイッチ SW1 を押したとき（*LOAD* = 0）に行われます．

表 4.10　リスト 4.14 のピン割り当て

記　号		ピン番号	備　考
入力	*CLK*	22	CLK1
	LOAD	34	SW1
	D-IN(0)	29	ロータリ SW の HEXSW3
	D-IN(1)	30	ロータリ SW の HEXSW2
	D-IN(2)	32	ロータリ SW の HEXSW1
	D-IN(3)	33	ロータリ SW の HEXSW0
出力	*Q*	61	LED4 の g

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity rei4_9 is
  Port ( D_IN : in std_logic_vector
  (3 downto 0);
         CLK : in std_logic;
         LOAD : in std_logic;
         Q : out std_logic );
end rei4_9;

architecture Behavioral of rei4_9 is
        signal WORK : std_logic_vector(3 downto 0) ;
        signal DD : std_logic_vector(21 downto 0) ;
        signal CE : std_logic ;
begin

 -- 分周回路     22 ビット
process (CLK)            ← 分周回路のカウンタ部
begin
  if CLK'event and CLK='1' then      ← クロックは CLK
      DD <= DD + '1';
  end if;
end process;

process (DD)             ← 分周回路の方形波発生部
begin
  if DD = "1111111111111111111111" then
     CE <= '1';
  else
     CE <= '0';
  end if;
end process ;

--  シフトレジスタ
process(CE)              ← シフトレジスタ部
        begin
        if (CE'event and CE = '1') then     ← クロックは CE
                 if (LOAD = '0') then       ← LOAD = 0 でデータ入力
                         WORK <= D_IN ;
                 else
                         WORK(0) <= WORK(3) ;
                         WORK(1) <= WORK(0) ;
                         WORK(2) <= WORK(1) ;
                         WORK(3) <= WORK(2) ;
                 end if ;
        end if ;
        end process ;
        Q <= WORK(3) ;           ← シリアル出力 Q への代入処理
end Behavioral;
```

リスト 4.14　シフトレジスタ（例題 4-9）の実習用コード

第4章

4.3 カウンタの設計

カウンタ（counter）は，特性方程式や励起表，カルノー図などを用いて設計することができます．しかし，VHDL を用いて動作記述すれば，任意の n 進カウンタをより簡単に設計することが可能となります．ここでは，VHDL によるカウンタの基本的な記述方法を実習しましょう．

▶4.3.1 同期式 n 進カウンタ

図 4.28 に，JK−FF を用いた同期式 5 進カウンタの回路などを示します．同期式カウンタは，クロック信号 CLK に同期して，すべての FF が一斉に動作します．また，n 進カウンタでは，有効なクロック信号 CLK が入力されるたびに，出力 Q_0 ～ Q_{n-1} の値は 1 ずつカウントアップし（アップカウンタの場合），n 個目のクロック信号ですべてリセットされます．そして，つぎのクロック信号から，再びカウントアップを繰り返します．n 進カウンタを構成するためには，$2^k \geq n$ を満たす k 個の FF を必要とします．図 4.28 で，5 進カウンタの動作を確認してください．

CLK	Q_2	Q_1	Q_0
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	0	0	0

(b) 動作表

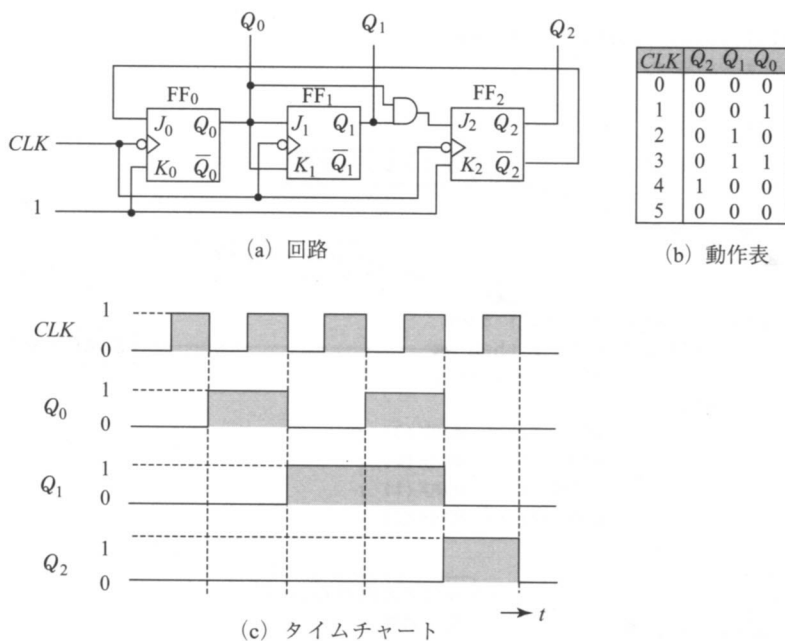



図 4.28　同期式 5 進カウンタ

特性方程式や励起表，カルノー図などを用いる場合には，2^n進カウンタが任意のn進カウンタよりも簡単に設計できますが，VHDLを用いれば，何進カウンタであってもほぼ同様のパターンで記述することができます．リスト4.15に，VHDLを用いた同期式8進カウンタ（2^n進カウンタ）のコードを示します．VHDLでは，出力Qに対して「Q <= Q + '1' ;」のような直接代入処理はできません．したがって，内部信号*WORK*を用いてカウントアップの処理を行っています．最後に，内部信号*WORK*を出力Qに代入する処理を記述することを忘れないでください．2^n進カウンタを記述する場合には，nビットの出力とそれに対応する内部信号を宣言します．

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity list15 is
  Port ( CLK : in std_logic;
         Q : out std_logic_vector(2 downto 0));
end list15;

architecture Behavioral of list15 is
        signal WORK : std_logic_vector(2 downto 0) ;
begin
process(CLK)
  begin
     if(CLK'event and CLK='0') then
                  WORK <= WORK + '1' ;
     end if;-
  end process;
  Q <= WORK ;
end Behavioral;
```


リスト4.15　同期式8進カウンタ（2^n進カウンタ）のコード

一方，任意のn進カウンタを記述する場合には，カウンタ処理を行う前にif文を用いた判定を行い，カウントが（$n-1$）になっていたら強制的にリセットをかけるようにします．リスト4.16に，同期式5進カウンタのコードを示しますのでリスト4.15と比較してください．

リスト4.17には，リセット機能の付いた7進カウンタのコードの一部（アーキテクチャ宣言部）を示します．非同期リセットと同期リセットの記述の違いを確認してください．両者では，process文のセンシティビティリストと信号*RESET*を判定するタイミングが異なります．

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity list16 is
    Port ( CLK : in std_logic;
           Q : out std_logic_vector(2 downto 0));
end list15;

architecture Behavioral of list16 is
       signal WORK : std_logic_vector(2 downto 0) ;    ← 内部信号 WORK の宣言
begin
process(CLK)
begin                                                  ← ネガティブエッジで動作
     if(CLK'event and CLK='0') then
              if(WORK = "100") then     ← (100)2 になっていたらリセット
                     WORK <= "000";
              else
                     WORK <= WORK + '1' ;  ← カウントアップ
            end if;
       end if;
end process;
Q <= WORK ;     ← 出力 Q への代入処理
end Behavioral;
```

リスト 4.16　同期式5進カウンタのコード

```
architecture Behavioral of list17a is
       signal WORK : std_logic_vector(2 downto 0) ;
begin
process(CLK,RESET)
begin
       if (RESET = '0') then
              WORK <= "000" ;
       elsif(CLK'event and CLK='0') then
              if(WORK = "110") then
                     WORK <= "000";

              else
                     WORK <= WORK + '1' ;
            end if;
       end if;
end process;
Q <= WORK ;
end Behavioral;
```

(a) 非同期リセットのアーキテクチャ宣言部

リスト 4.17　リセット付き7進カウンタのコードの一部 (1)

```
architecture Behavioral of list17b is
        signal WORK : std_logic_vector(2 downto 0) ;
begin
process(CLK)
begin
        if (CLK'event and CLK='0') then
                if (RESET = '0') then
                        WORK <= "000" ;
                elsif(WORK = "110") then
                        WORK <= "000";
                else
                        WORK <= WORK + '1' ;
              end if;
        end if;
end process;
Q <= WORK ;
end Behavioral;
```

(b) 同期リセットのアーキテクチャ宣言部

リスト 4.17　リセット付き 7 進カウンタのコードの一部（2）

例題 4－10

図 4.29 に示す同期式 10 進カウンタと同様の動作をする回路を VHDL で記述しなさい．ただし，端子 ***RESET*** は，同期リセットとすること．

<動作表>

CLK	Q_3	Q_2	Q_1	Q_0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	0	0	0	0

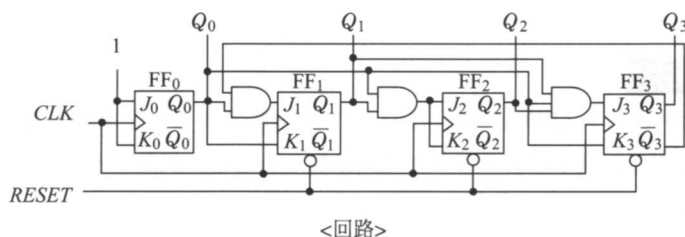



図 4.29　同期式 10 進カウンタ

解答例

リスト 4.17（b）を参考にコードを記述しましょう．10 進カウンタでは，4 個の FF が必要になります．リスト 4.18 に，同期式 10 進カウンタのコードを示します．

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity rei4_10 is
    Port ( CLK : in std_logic;
         RESET : in std_logic;
             Q : out std_logic_vector(3 downto 0));
end rei4_10;
```

4 ビット出力の宣言

```
architecture Behavioral of rei4_10 is
        signal WORK : std_logic_vector(3 downto 0) ;
begin
process(CLK)
begin
        if (CLK'event and CLK='1') then        ← CLKのポジティブエッジで動作
                if (RESET = '0') then          ← 同期リセット
                    WORK <= "0000" ;
                elsif(WORK = "1001") then      ← (1001)2になっていたらリセット
                    WORK <= "0000";
                else
                    WORK <= WORK + '1' ;       ← カウントアップ
          end if;
        end if;
end process;
Q <= WORK ;                                    ← 出力Qへの代入処理
end Behavioral;
```

リスト 4.18　同期式 10 進カウンタのコード

▶4.3.2　カウンタの応用

例題 4－10 で設計した 10 進カウンタを HDL トレーナーのクロック 4 MHz で動作させると高速すぎて人の目では確認できません（111 ページ参照）．したがって，例題 4－11 で動作確認を行いましょう．

例題 4－11

7 セグメント LED を「0」～「9」まで順次表示することを繰り返す回路を VHDL で記述しなさい．ただし，表示の切り替わり時間は約 1 秒とし，同期リセット端子 *RESET* を備えること．

解答例

図 4.30 に，HDL トレーナーを用いた 10 秒をカウント表示する回路の構成を示します．

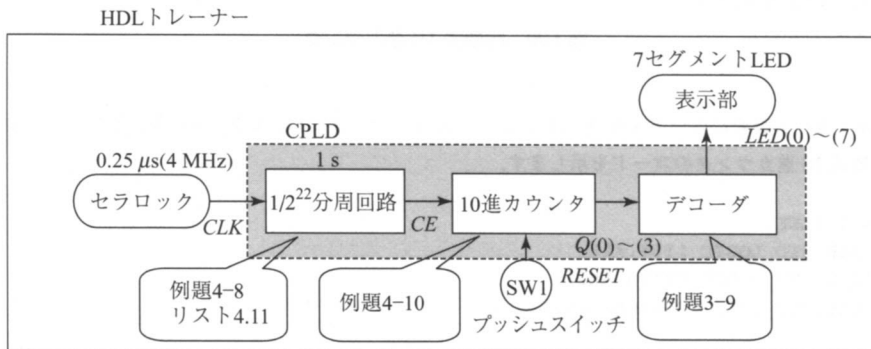



図 4.30　10 秒をカウント表示する回路の構成

主要な回路は，分周回路，10 進カウンタ，デコーダの三つです．これらはすべて，つぎの例題で学習しています．

・分周回路 → 例題 4-8（リスト 4.11）

・10 進カウンタ → 例題 4-10（リスト 4.18）

・デコーダ → 例題 3-9（リスト 3.25）

したがって，これらの回路を組合せれば 10 秒をカウント表示する回路を構成することができます．リスト 4.19 にコードの記述例を示しますので，表 4.11 に示すピン割り当てによって実習を行いましょう．リスト 4.19 を論理合成すると，「WARNING（:Xst:737 - Found 8-bit latch for signal <LED>.)」が出ますが，これはラッチ回路があることを示すメッセージなので問題ありません．

ここでは，実機（HDL トレーナー）による動作確認を行いましたが，シフトレジスタやカウンタなどの順序回路では，シミュレータソフトウェアを用いてシミュレーションを行うと動作確認が容易になります．シミュレーションについては，第 6 章で説明します．

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity rei4_11 is
  Port (CLK : in std_logic;
        RESET : in std_logic;
        LED : out std_logic_vector(7 downto 0));
end rei4_11;

architecture Behavioral of rei4_11 is
        signal DD : std_logic_vector(21 downto 0) ;
        signal CE : std_logic ;
        signal WORK : std_logic_vector(3 downto 0) ;
begin

-- 分周回路    22 ビット
process (CLK)
begin
  if CLK'event and CLK='1' then
      DD <= DD + '1';
  end if;
end process;

process (DD)
begin
  if DD = "1111111111111111111111" then
     CE <= '1';
  else
     CE <= '0';
  end if;
end process;

-- カウンタ回路
process(CE)
begin
        if (CE'event and CE='1') then
                if (RESET = '0') then
                        WORK <= "0000" ;
                elsif(WORK = "1001") then
                        WORK <= "0000";
                else
                        WORK <= WORK + '1' ;
            end if;
        end if;
end process;

--デコーダ回路
process(WORK)
begin
        case WORK is
                when "0000" => LED <= "00000011" ;
                when "0001" => LED <= "10011111" ;
                when "0010" => LED <= "00100101" ;
                when "0011" => LED <= "00001101" ;
                when "0100" => LED <= "10011001" ;
                when "0101" => LED <= "01001001" ;
                when "0110" => LED <= "01000001" ;
                when "0111" => LED <= "00011011" ;
                when "1000" => LED <= "00000001" ;
                when "1001" => LED <= "00001001" ;
                when others => null ;
        end case;
end process;
end Behavioral;
```

リスト 4.19 10 秒をカウント表示する回路のコード

第4章

表 4.11　例題 4-11 のピン割り当て

	記号	ピン番号	備考	
入力	*CLK*	22	CLK1	
	RESET	34	SW1	
出力	*LED*(0)	50	LED4 (LEDD)	dp
	LED(1)	61		g
	LED(2)	60		f
	LED(3)	58		e
	LED(4)	56		d
	LED(5)	55		c
	LED(6)	53		b
	LED(7)	52		a

例題 4-12

4 MHz のクロック信号をおよそ 4 kHz に分周する回路を VHDL で記述しなさい．そして，分周した信号を用いて HDL トレーナーのブザーを鳴らしなさい．

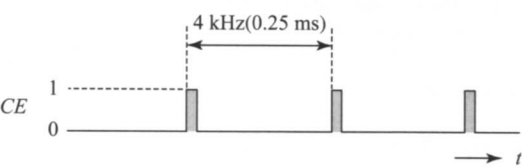
解答例

HDL トレーナーに搭載されているブザーは，連続した電圧を加えただけでは鳴らすことはできません．人が聞くことのできる周波数（およそ 20 Hz ～ 20 kHz）の信号を加える必要があります．したがって，HDL トレーナーのセラロックが発振する 4 MHz の方形波を分周してからブザーに加えます．たとえば，周波数 4 kHz の方形波を発生するには，次式から 1/1000 に分周すればよいことがわかります．

$$4\,\mathrm{MHz} \div x = 4\,\mathrm{kHz}$$

$$x = (4 \times 10^6) \div (4 \times 10^3) = 1000$$

$2^{10} = 1024 \fallingdotseq 1000$ と考えて，10 ビットで分周を行うことにします．これまでの記述法（リスト 4.11 など）によって，10 ビットの分周回路を記述するとリスト 4.20 に示すコード（process 文）になります．また，このコードで発生する出力信号 *CE* の波形を図 4.31 に示します．

```
process (CLK)
begin
   if CLK'event and CLK='1' then
        DD <= DD + '1';
   end if;
end process;

process (DD)
begin
   if DD = "1111111111" then
      CE <= '1';
   else
      CE <= '0';
   end if;
end process;
```

リスト 4.20　10 ビット分周回路のコード（その 1）

4 kHz(0.25 ms)
CE
1
0
t

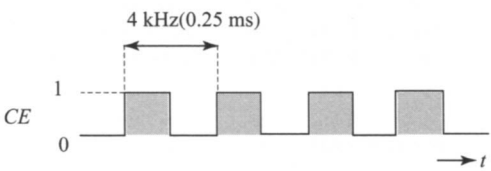
図 4.31　*CE* の波形（リスト 4.20）

図4.31では，波形が0と1である時間が同じではありません．これまでの実習では，ポジティブエッジまたはネガティブエッジを得るために分周していたので問題はありませんでした．しかし，この例題では，0と1が同じ時間で変化する波形を得る必要があります．

リスト4.21に，0と1が同じ時間で変化する波形を得る分周回路のコード，図4.32に*CE*の波形を示します．リスト4.21では，分周回路で$2^{10}/2$のタイミングで，*CE*を切り替えていることに注目してください．

4 kHz(0.25 ms)

図4.32 *CE*の波形（リスト4.21）

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity rei4_12b is
    Port (CLK : in std_logic;
          BZ : out std_logic);
end rei4_12b;

architecture Behavioral of rei4_12b is
       signal DD : std_logic_vector(9 downto 0) ;
       signal CE : std_logic ;
begin

-- 分周回路
process (CLK)
begin
   if CLK'event and CLK='1' then
        DD <= DD + '1';
   end if;
end process;

process (DD)
begin
   if DD = "0111111111" then
      CE <= '1';
   elsif DD = "1111111111" then
      CE <= '0';
   end if;
end process;
BZ <= CE ;
end Behavioral;
```

半分のカウント時間ごとに*CE*を切り替えている

リスト4.21 10ビット分周回路のコード（その2）

表4.12に示すピン割り当てによって実習を行いましょう．リスト4.20とリスト4.21のコードをダウンロードして，ブザー音を比べてみましょう．また，ブザーに加える信号の周波数を変化させる実習も行うとよいでしょう．

表4.12 例題4-12のピン割り当て

記号		ピン番号	備考
入力	*CLK*	22	CLK1
出力	*BZ*	40	BZOUT

▶ 演習問題 4

4.1 リスト4.22は，図4.33に示す *JK*−FFをVHDLで記述したコードである．①〜⑩に適切な語句を埋めてコードを完成しなさい．ただし，*JK*−FFの ***RESET*** 端子は，同期式リセットとする．

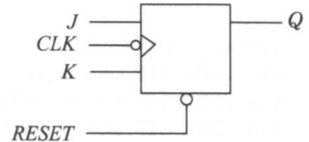
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity syo4_1 is
   Port ( J,K,CLK,RESET : in std_logic;
          Q : out std_logic);
end syo4_1;

architecture Behavioral of syo4_1 is
        signal INPUT : std_logic_vector(1 downto 0) ;
        signal WORK : std_logic    ;
begin
        INPUT <= (①) ;
        process(②)
        begin
                if (CLK'event and CLK = ③) then
                         if (④ = ⑤) then
                            WORK <= '0' ;
                            else
                                 ⑥ INPUT is
                                         when "01" => WORK <= '0' ;
                                         when "10" => WORK <= '1' ;
                                         when "11" => WORK <=  ⑦  ;
                                         when ⑧ => null ;
                                 ⑨ ;
                         end if ;
                end if ;
        end process ;
        Q <= ⑩ ;
end Behavioral;
```


図4.33　*JK*−FF

リスト4.22　*JK*−FFのコード

4.2 同期式4進カウンタをVHDLで記述しなさい．ただし，クロック信号のポジティブエッジで動作し，同期リセット端子（負論理）をもつこととする．

4.3 同期式9進カウンタをVHDLで記述しなさい．ただし，クロック信号のネガティブエッジで動作し，同期リセット端子（負論理）をもつこととする．

4.4 例題4−11（122ページ）で設計した10秒をカウント表示する回路において，カウンタ部をダウンカウンタに変更して，7セグメントLEDが0→9→8→〜→1→0と順次表示するコードを記述しなさい．ただし，カウントダウンの記述には演算子「−」を使用すること．

第5章
階層設計

これまでの実習では，分周回路，カウンタ回路，デコーダ回路などで構成されるディジタル回路を一つのコードとして記述してきました．つまり，一つのアーキテクチャ宣言部に各回路を記述していました．しかし，複雑なディジタル回路では，各部を分割して記述した方が構造を理解しやすくなり，また各種のトラブルを減少させることにもつながります．さらに，何度も使用する回路を独立したコードとして記述しておき，必要なときにそのコードを参照するようにすれば，何度も同じコードを記述する必要はなくなります．このように，分割した各回路をまとめ上げることで，全体の回路を構成するような設計方法を階層設計とよびます．この章では，階層設計の基本的な考え方と実際の記述法を理解しましょう．

5.1 階層設計の基礎

複雑なディジタル回路の設計では，階層設計の考え方が不可欠になってきます．ここでは，階層設計の基本的な考え方と実際の記述パターンを理解しましょう．また，7セグメントLEDを2桁使用する100秒カウンタや，電子サイコロなどの応用回路についての実習を行いましょう．

5.1.1 階層設計とは

全減算器（FS）の動作記述については，例題3-6（85ページ）で実習しました．一方，全減算器は，図5.1に示すように2個の半減算器（HS）によって構成することもできます．ここでは，図5.1を例にして階層設計について考えましょう．

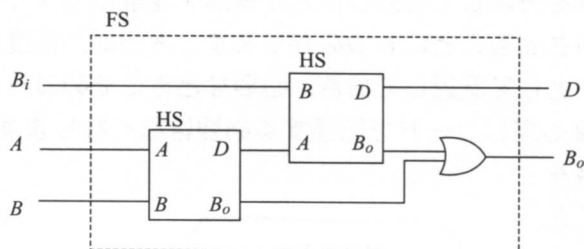



図5.1　半減算器（HS）を用いた全減算器（FS）の構成

図5.1では，同じ働きをする半減算器が2個使用されています．VHDLを用いてコードを記述する場合，各半減算器を個別に記述することもできますが，それでは効率がよいとはいえません．できれば，一箇所に記述した半減算器をうまく利用して全減算器の回路を構成したいものです．また，図5.1では，同じ機能（半減算器）を複数箇所で使用する場合を例に上げましたが，異なる機能である場合でも，各機能を個別に記述して，それらをまとめて全体を構成すると考え方が簡単になります．さらに，機能ごとに回路を考える方がエラーの発見が容易となりトラブルの減少にもつながります．複数の設計者が共同で大規模な回路を設計する場合にも，各機能を分担することが必要になります．このような記述を実現するのが階層設計の考え方です．

階層設計は，プログラミング言語におけるモジュール化やサブルーチンに似た考え方であるととらえることができます．しかし，サブルーチンでは同じメモリブロックにあるサブルーチンプログラムが繰り返し実行されますが，VHDLの階層設計で記述した機能は使用される回数分の回路が実際にCPLD/FPGA内に構成される点が異なります．

▶5.1.2 階層設計の方法

階層設計では，コンポーネント（component）宣言とポートマップ（port map）宣言を使用した記述が基本となります．図 5.2 に，階層設計記述の流れを示します．

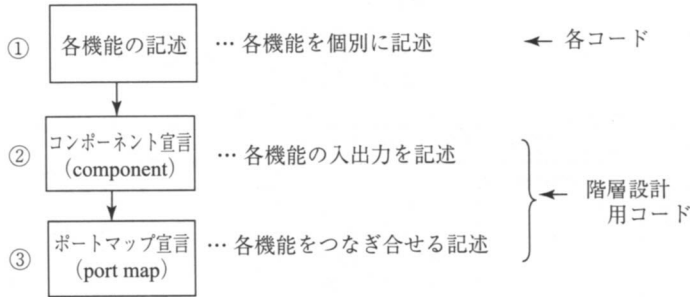



図 5.2 階層設計記述の流れ

初めに，各機能のコードを個別に記述しておきます．そして，階層設計用のコードには，コンポーネント宣言によって別に記述した各機能の入出力端子に関する情報を記述し，ポートマップ宣言によって各機能をつなぎ合わせる記述を行います．図 5.3 に，階層設計記述のイメージを示します．

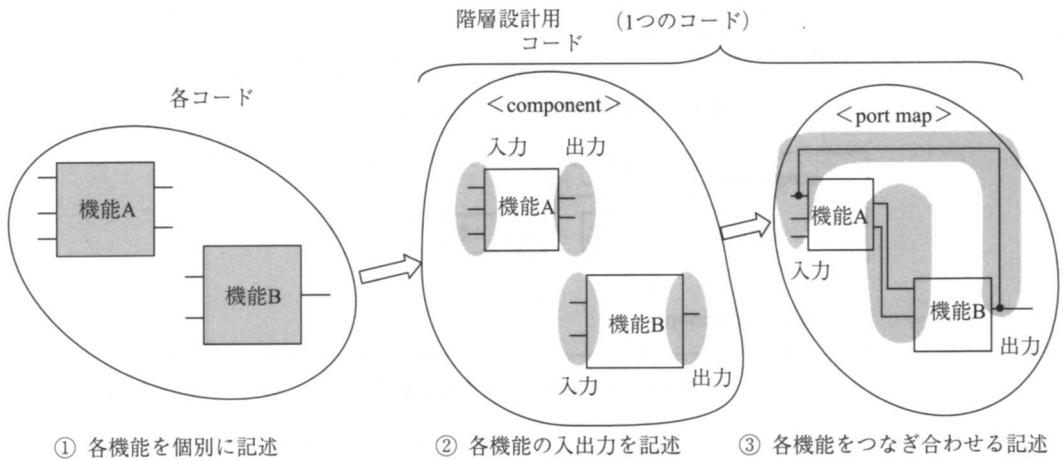



図 5.3 階層設計記述のイメージ

図 5.4 に階層設計用コードの記述パターン，図 5.5，5.6 にコンポーネント宣言とポートマップ宣言の書式を示します．コンポーネント宣言とポートマップ宣言は，どちらもアーキテクチャ宣言内に記述します．コンポーネント宣言は信号宣言部，ポートマップ宣言は機能宣言部に記述します（62 ページ図 3.7 参照）．

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity fs is
      Port (A: in std_logic;
            B: in std_logic;
            B_I : in std_logic;
            D : out std_logic;
            B_O : out std_logic);
end fs;

architecture Behavioral of fs is
  component hs
        Port (A : in std_logic;
              B : in std_logic;
              D : out std_logic;
              BO : out std_logic);
  end component;
      signal HS1_D, HS1_BO, HS2_BO : std_logic ;
begin
  C1:hs
        port map(A =>, B =>B, D => HS1_D, BO => HS1_BO);
  C2:hs
      port map(A => HS1_D,B => B_I, D => D, BO => HS2_BO);
  B_0 <= HS1_BO or HS2_BO ;
end Behavioral;
```


図5.4　階層設計用コードの記述パターン

```
component [コンポーネント名]
   Port ( [ポート名] : [モード型]  [データ型] );

end component ;
```

図5.5　コンポーネント宣言の書式

```
[ラベル] : [コンポーネント名]

   port map ( [コンポーネントのポート名] => [代入先] );
```

図5.6　ポートマップ宣言の書式

コンポーネント宣言は，各機能を記述したコードのエンティティ宣言をコピーして利用するとよいでしょう．また，ポートマップ宣言では，代入の方向を間違えないようにしてください．

図5.7に示す全減算器を例にして階層設計の記述パターンを学習しましょう．図5.2に示した流れに沿って作業を進めていきましょう．初めは，各機能を個別に記述する作業，つまり半減算器の記述からです．

図5.7 半減算器を用いた全減算器の構成

① 各機能の記述

半減算器を記述します．図5.8に半減算器各部の信号，リスト5.1にコードを示します（例題3-5，84ページ参照）．

図5.8 半減算器各部の信号

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity hs is
  Port ( A : in std_logic;
         B : in std_logic;
         D : out std_logic;
         BO : out std_logic);
end hs;

architecture Behavioral of hs is
        signal X, Y:std_logic ;
begin
        X <= A and (not B) ;
        Y <= B and (not A) ;
        D <= X or Y ;
        BO <= Y ;
end Behavioral;
```

リスト5.1 半減算器のコード

これまでの実習と同様にISE WebPACKを起動します．そして，ツールバーの「File」→「New Project」を選択し（図5.9参照），たとえば「sample1」という名でプロジェクトを作成します．続いて，ツールバーの「Project」→「New Source」を選択し（図5.10参照），「VHDL Module」のファイル名をたとえば「hs」として新規コードを記述するウインドウを開きます．この際，入力はAとB，出力はDとBO，すべて「std_logic型」です．

第5章

図 5.9　「File」→「New Project」を選択

図 5.10　「Project」→「New Source」を選択

リスト 5.1 に示したコードを入力したら，論理合成「Synthesize－XST」を実行してエラーのないことを確認します（図 5.11）.

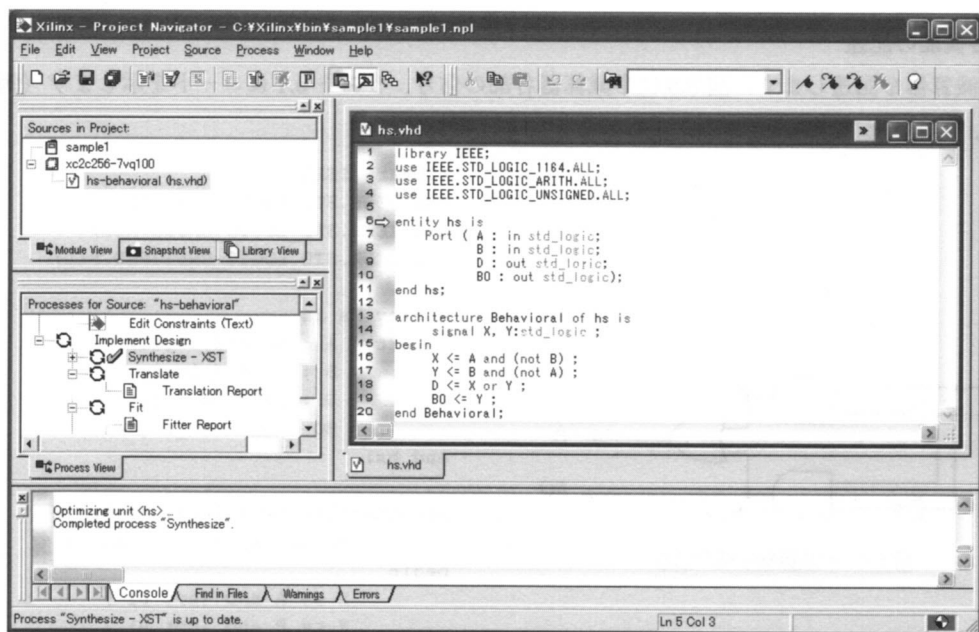



図 5.11　半減算器の論理合成「Synthesize－XST」を実行

② コンポーネント宣言

続いて，階層設計用のコードを記述します．図 5.12 に，設計する全減算器各部の信号を示します．

ツールバーの「Project」→「New Source」を選択し（図 5.13 参照），「VHDL Module」のファイル名をたとえば「fs」として新規コードを記述するウインドウを開きます（図 5.14 参照）．この際，入力は A，B，B_I，出力は D と B_O，すべて「std_logic 型」です．

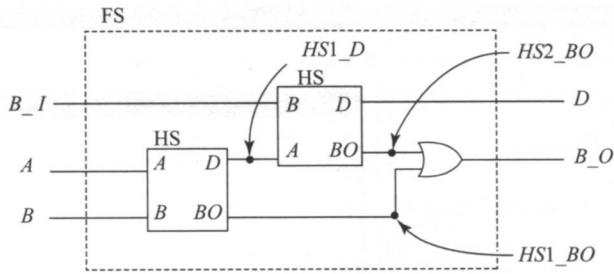



図 5.12 全減算器各部の信号

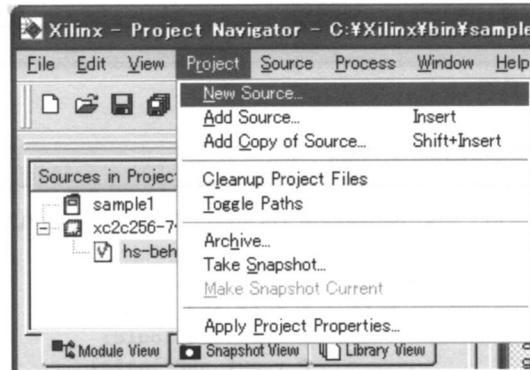



図 5.13 「Project」→「New Source」を選択

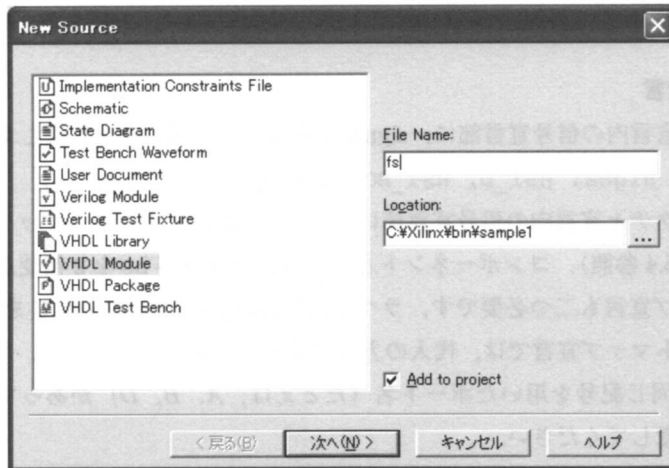



図 5.14 「VHDL Module」のファイル名を「fs」とする

図 5.15 に示すように，「Sources in Project」ウインドウには，先ほど作成した「hs-behavioral (hs.vhd)」コードに加えて，「fs-behavioral (fs.vhd)」コードが追加されているはずです．アーキテクチャ宣言内の信号宣言部に，リスト 5.2 に示すコンポーネント宣言を記述します（130 ページ図 5.4 参照）．

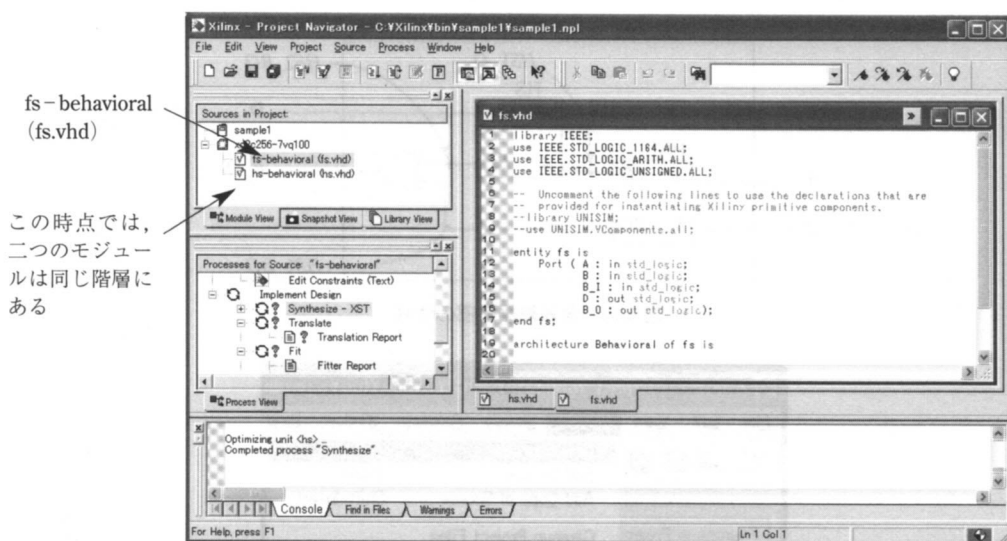



図 5.15　「fs－behavioral（fs.vhd）」コードが追加された

```
component hs
        Port ( A : in std_logic;
               B : in std_logic;
               D : out std_logic;
               BO : out std_logic);
end component;
```

リスト 5.2　コンポーネント宣言

③ ポートマップ宣言

アーキテクチャ宣言内の信号宣言部に，signal 文を用いて内部信号を宣言します．

```
signal HS1_D, HS1_BO, HS2_BO : std_logic ;
```

続いて，アーキテクチャ宣言内の信号宣言部に，リスト 5.3 に示すポートマップ宣言を記述します（130 ページ図 5.4 参照）．コンポーネントとしては，半減算器 hs を 2 個使用しますので，記述するポートマップ宣言も二つ必要です．ラベルは，component の頭文字 C を用いて，C1，C2 としました．ポートマップ宣言では，代入の方向によってコンポーネントと，代入先を区別します．したがって，同じ記号を用いたポート名（たとえば，A，B，D）があってもかまいませんが，記述順序に注意してください．

```
C1:hs
    port map(A => A, B => B, D => HS1_D, BO => HS1_BO) ;

C2:hs
    port map(A => HS1_D, B => B_I, D => D, BO => HS2_BO) ;
```

リスト 5.3　ポートマップ宣言

アーキテクチャ宣言の機能宣言部に，全減算器の出力 B_O を得るための OR 回路を記述します．

```
B_O <= HS1_BO or HS2_BO ;
```

階層設計用コードの記述が終われば，ツールバーの「File」→「Save」（または保存ボタン）を選択してコードを保存します．半減算器のコード「hs-behavioral（fs.vhd）」は，全減算器のコード「fs-behavioral（fs.vhd）」（階層設計用コード）の下位のモジュールになりますので，タイトルが一段下がって表示されているはずです（図5.16参照）．

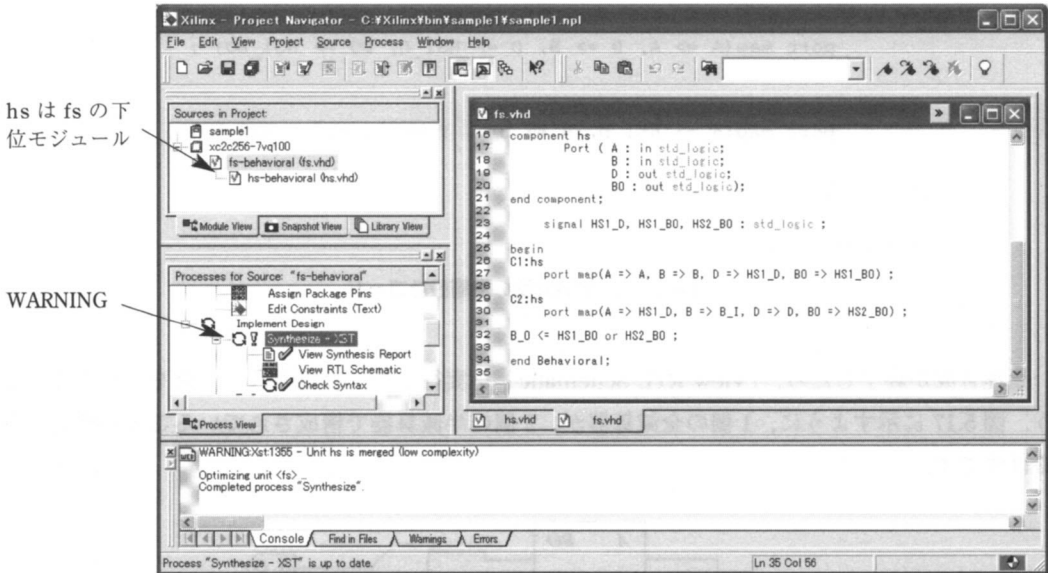



図5.16　階層設計用のコード記述ウインドウ

リスト5.4に示したコードを入力したら，「Sources in Project」ウインドウで「fs-behavioral（fs.vhd）」が選択されていることを確認して，論理合成「Synthesize-XST」を実行します．この場合には，「WARNING:Xst:1355 - Unit hs is merged（low complexity）」という警告が出ますが，これはコンポーネントhsを取り込んだというメッセージですので問題ありません（図5.16参照）．

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity fs is
    Port ( A : in std_logic;
           B : in std_logic;
           B_I : in std_logic;
           D : out std_logic;
           B_O : out std_logic);
end fs;

architecture Behavioral of fs is

component hs
```

```
          Port ( A : in std_logic;
             B : in std_logic;
             D : out std_logic;
             BO : out std_logic);
end component;

         signal HS1_D, HS1_BO, HS2_BO : std_logic ;

begin
C1:hs
         port map(A => A, B => B, D => HS1_D, BO => HS1_BO) ;

C2:hs
         port map(A => HS1_D, B => B_I, D => D, BO => HS2_BO) ;

B_O <= HS1_BO or HS2_BO ;

end Behavioral;
```

リスト 5.4　全減算器の階層設計コード

論理合成が終了したら，「View RTL Schematic」を実行して，合成された回路をみてみましょう．図 5.17 に示すように，1 個の全減算器が，2 個の半減算器で構成されていることが確認できるはずです．

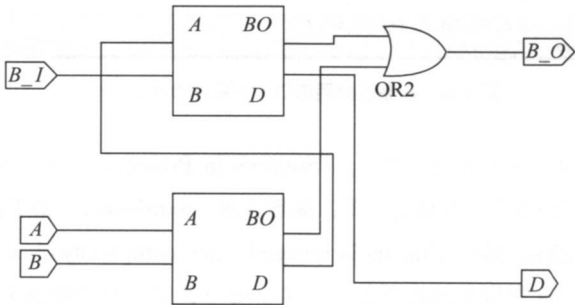



図 5.17　合成された回路

つぎに，表 5.1 に示すピン割り当てを行い，HDL トレーナーにファイル「fs.jed」をダウンロードして実習を行いましょう．念のため，表 5.2 に全減算器の真理値表を示します．

表 5.1　全減算器のピン割り当て

記　号		ピン番号	備　考
入力	*A*	34	SW1
	B	35	SW2
	B_I	36	SW3
出力	*D*	74	LED3 の g
	B_O	61	LED4 の g

表 5.2 全減算器の真理値表

A	B	B_I	D	B_O
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

ポートマップ宣言の書式は図 5.6（130 ページ）に示しましたが，リスト 5.5 に示す記述法もあります．この記述法では，コンポーネント宣言に記述したポート名（A，B，D，BO）と同じ順で，ポートマップ宣言に記述した代入先への代入処理が行われます．

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity fs is
    Port ( A : in std_logic;
           B : in std_logic;
           B_I : in std_logic;
           D : out std_logic;
           B_O : out std_logic);
end fs;

architecture Behavioral of fs is

component hs
        Port ( A : in std_logic;
               B : in std_logic;
               D : out std_logic;
               BO : out std_logic);
end component;

       signal HS1_D, HS1_BO, HS2_BO : std_logic ;

begin
C1:hs
       port map(A , B, HS1_D, HS1_BO) ;

C2:hs
       port map(HS1_D, B_I, D, HS2_BO) ;

B_O <= HS1_BO or HS2_BO ;

end Behavioral;
```


リスト 5.5 ポートマップ宣言の記述法

第5章

5.2 階層設計の実習

ゲートとFFが記述できるようになり，階層設計が理解できれば，VHDLを用いて多くのディジタル回路を設計することが可能になります．ここでは，階層設計の理解を深めるためにいくつかの例題を実習しましょう．

5.2.1 10秒カウンタの設計

例題4-11（122ページ）で実習した10秒カウンタ回路を，ここでは階層的に設計してみましょう．

例題5-1

7セグメントLEDを「0」～「9」まで順次表示することを繰り返す回路を階層的に設計しなさい（図5.18参照）．ただし，表示の切り替わり時間は約1秒とし，同期リセット端子 ***RESET*** を備えること．

第5章

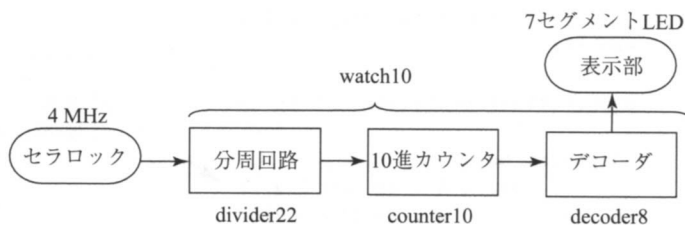



図5.18　10秒カウンタの構成

解答例

分周回路，10進カウンタ，デコーダを独立したコード（モジュール）として記述し，階層設計用コードでまとめましょう．各コードの名前は，階層設計用を「watch10」，分周回路を「divider22」，カウンタを「counter10」，デコーダを「decoder8」とします．これらのコードの階層構造を図5.19に示します．「watch10」が最上位に位置し，その下位に三つのコードが並んだ形になります．

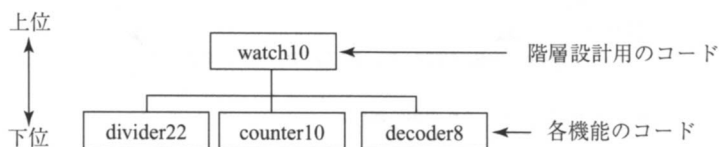



図5.19　10秒カウンタの階層構造

図5.20に，設計する回路の各部の信号を示します．

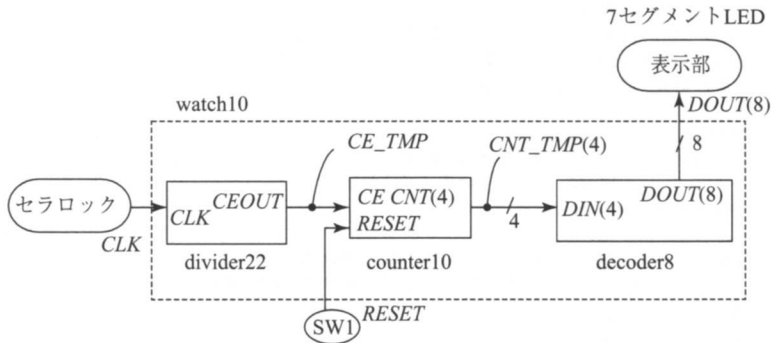



図 5.20 10 秒カウンタの各部の信号

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity watch10 is
    Port ( CLK : in std_logic;
           RESET : in std_logic;
           DOUT : out std_logic_vector(7 downto 0));
end watch10;

architecture Behavioral of watch10 is

component divider22
    Port (CLK : in std_logic;
          CEOUT : out std_logic);
end component;

component counter10
    Port (CE : in std_logic;
          RESET : in std_logic;
          CNT : out std_logic_vector(3 downto 0));
end component;

component decoder8
    Port (DIN : in std_logic_vector(3 downto 0);
          DOUT : out std_logic_vector(7 downto 0));
end component;

       signal CE_TMP : std_logic ;
       signal CNT_TMP : std_logic_vector(3 downto 0) ;

begin
C1:divider22
       port map(CLK => CLK, CEOUT => CE_TMP) ;
C2:counter10
       port map(CE => CE_TMP, RESET => RESET, CNT => CNT_TMP) ;
C3:decoder8
       port map(DIN => CNT_TMP, DOUT => DOUT) ;

end Behavioral;
```

リスト 5.6 10 秒カウンタ階層設計用（watch10）のコード

第5章

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity divider22 is
   Port (CLK : in std_logic;
         CEOUT : out std_logic);
end divider22;

architecture Behavioral of divider22 is
        signal DD : std_logic_vector(21 downto 0) ;
begin
process (CLK)
begin
   if CLK'event and CLK='1' then
       DD <= DD + '1';
   end if;
end process;

process (DD)
begin
   if DD = "1111111111111111111111" then
      CEOUT <= '1';
   else
      CEOUT <= '0';
   end if;
end process;

end Behavioral;
```

リスト 5.7　分周回路（divider22）

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity counter10 is
   Port (CE : in std_logic;
         RESET : in std_logic;
         CNT : out std_logic_vector(3 downto 0));
end counter10;

architecture Behavioral of counter10 is
        signal WORK : std_logic_vector(3 downto 0) ;
begin
process(CE)
begin
        if (CE'event and CE='1') then
                 if (RESET = '0') then
                          WORK <= "0000" ;
                 elsif(WORK = "1001") then
                          WORK <= "0000";
                 else
                          WORK <= WORK + '1' ;
        end if;
    end if;
end process;

CNT <= WORK ;

end Behavioral;
```

リスト 5.8　カウンタ（counter10）のコード

第5章

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity decoder8 is
    Port (DIN : in std_logic_vector(3 downto 0);
          DOUT : out std_logic_vector(7 downto 0));
end decoder8;

architecture Behavioral of decoder8 is

begin
process(DIN)
begin
        case DIN is
                when "0000" => DOUT <= "00000011" ;
                when "0001" => DOUT <= "10011111" ;
                when "0010" => DOUT <= "00100101" ;
                when "0011" => DOUT <= "00001101" ;
                when "0100" => DOUT <= "10011001" ;
                when "0101" => DOUT <= "01001001" ;
                when "0110" => DOUT <= "01000001" ;
                when "0111" => DOUT <= "00011011" ;
                when "1000" => DOUT <= "00000001" ;
                when "1001" => DOUT <= "00001001" ;
                when others => null ;
        end case;
end process;

end Behavioral;
```

リスト 5.9 デコーダ（decorer8）のコード

リスト 5.6 に階層設計用のコード，リスト 5.7，5.8，5.9 に下位に位置する三つの回路のコードを示します．下位のコードは，123 ページのリスト 4.19 を三つに分割したものと考えることができます．

まず，ツールバーの「Project」→「New Source」を選択して下位に位置する三つのコードをおのおの記述します．図 5.21 に，三つのコードを記述した後の「Sources in Project」ウインドウを示します．

図 5.21 で各コードをクリックして選択した後，論理合成「Synthesize - XST」を実行します．デコーダ（decorder8）で，「WARNING : Xst : 737 - Found 8-bit latch for signal < DOUT >.」という警告が出るかもしれませんが，これはラッチ回路が見つかったという確認メッセージですので問題ありません．

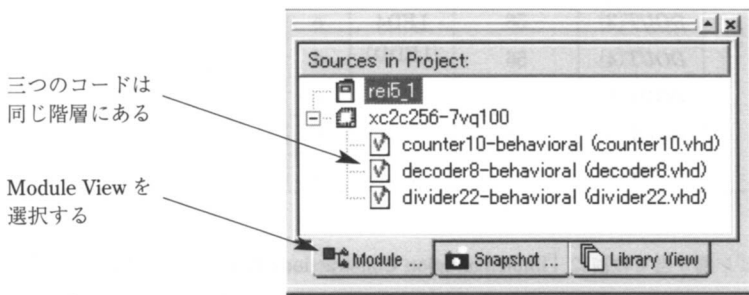



図 5.21 「Sources in Project」ウインドウ

第5章

続いて，階層設計用コード（watch10）を記述すると，図 5.22 に示すように「Sources in Project」ウインドウに各コードの階層構造が表示されます．

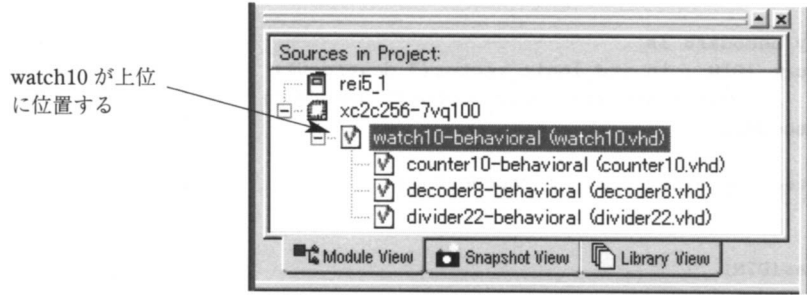



図 5.22　各コードの階層構造

「Sources in Project」ウインドウで「watch10-behavional（watch10.vhd）」を選択した後，「Assign Package Pins」をダブルクリックして（図 5.23 参照），表 5.3 に示すピン割り当てを行います．

図 5.23　「Assign Package Pins」をダブルクリックする

表 5.3　例題 5-1 のピン割り当て

記　号		ピン番号	備　考	
入力	*CLK*	22	CLK1	
	RESET	34	SW1	
出力	*DOUT*(0)	50	LED4 (LEDD)	dp
	DOUT(1)	61		g
	DOUT(2)	60		f
	DOUT(3)	58		e
	DOUT(4)	56		d
	DOUT(5)	55		c
	DOUT(6)	53		b
	DOUT(7)	52		a

なお，47 ページの説明では，ピン割り当ての前に「Implemantation Constractions File」を作成しました．しかし，このファイルの作成前に「Assign Package Pins」をダブルクリックするとファイル作成を自動的に作成する確認メッセージが現れます（図 5.24 参照）．

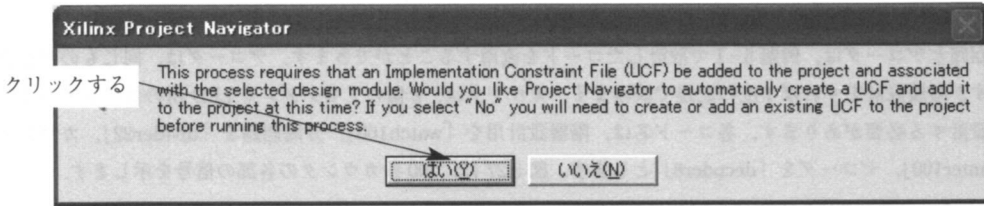



図 5.24 「Implemantation Constractions File」自動作成メッセージ

図 5.25 に，ピン割り当て用ファイル「watch10.ucf」を作成した後の「Sources in Project」ウインドウを示します．

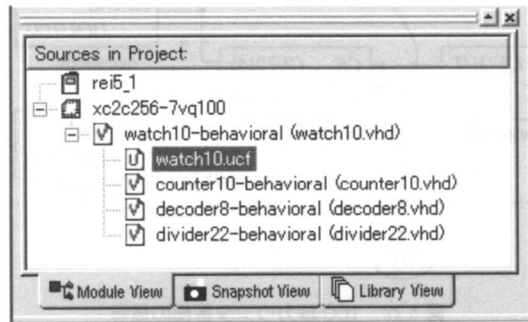



図 5.25 「watch10.ucf」を作成した

ピン割り当てを終えたらば，HDL トレーナーに「watch10.jed」ファイルをダウンロードして動作実習を行いましょう．

▶5.2.2 100 秒カウンタの設計

例題 5-1 では，1 桁の 7 セグメント LED を使用した 10 秒カウンタを設計しました．ここでは，2 桁の 7 セグメント LED を使用した 100 秒カウンタを階層的に設計してみましょう．

例題 5-2

2 桁の 7 セグメント LED を「00」〜「99」まで順次表示することを繰り返す回路を階層的に設計しなさい（図 5.26 参照）．ただし，表示の切り替わり時間は約 1 秒とし，同期リセット端子 *RESET* を備えること．

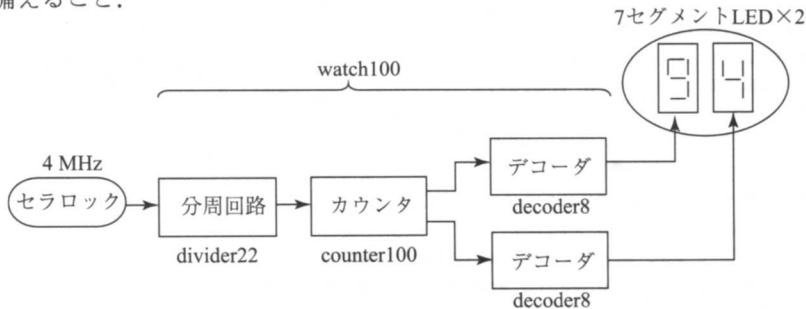



図 5.26 100 秒カウンタの構成

解答例

分周回路とデコーダは，例題5-1で設計したコードを流用することができます．デコーダは，同じものが2個必要ですが階層設計をすれば1度の記述で済みます．カウンタは，2個の7セグメントLED用の信号を出力するように設計する必要があります．各コード名は，階層設計用を「watch100」，分周回路を「divider22」，カウンタを「counter100」，デコーダを「decoder8」とします．図5.27に，100秒カウンタの各部の信号を示します．

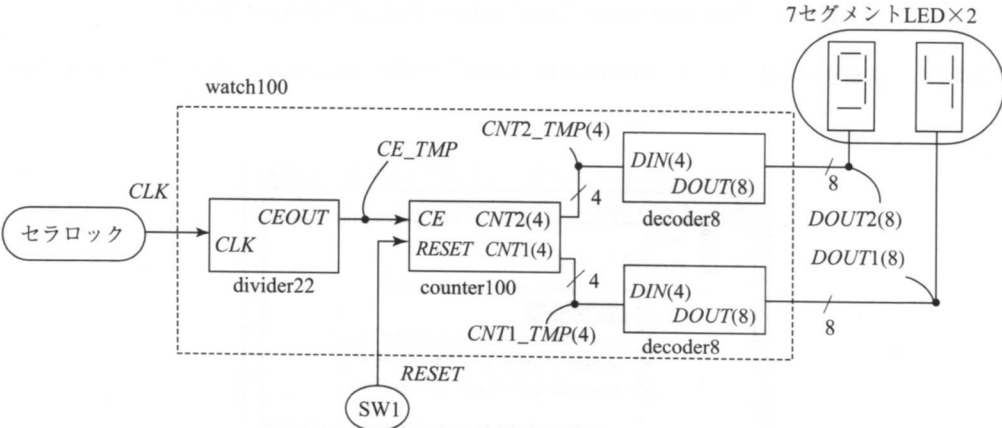



図5.27　100秒カウンタ各部の信号

リスト5.10に階層設計用コード，リスト5.11にカウンタのコードを示します．分周回路（divider22）はリスト5.7，デコーダ（decoder8）はリスト5.9と同じです．

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity watch100 is
    Port ( CLK : in std_logic;
           RESET : in std_logic;
           DOUT1 : out std_logic_vector(7 downto 0) ;
           DOUT2 : out std_logic_vector(7 downto 0));
end watch100;

architecture Behavioral of watch100 is

component divider22
    Port (CLK : in std_logic;
          CEOUT : out std_logic);
end component;

component counter100
    Port (CE : in std_logic;
          RESET : in std_logic;
          CNT1 : out std_logic_vector(3 downto 0) ;
          CNT2 : out std_logic_vector(3 downto 0));
end component;
```

分周回路のコンポーネント宣言

カウンタのコンポーネント宣言

```
component decoder8  ← デコーダのコンポーネント宣言
    Port (DIN : in std_logic_vector(3 downto 0);
       DOUT : out std_logic_vector(7 downto 0));
end component;
                                              ← 内部信号の宣言
       signal CE_TMP : std_logic ;
       signal CNT1_TMP : std_logic_vector(3 downto 0) ;
       signal CNT2_TMP : std_logic_vector(3 downto 0) ;
begin
C1:divider22
    port map(CLK => CLK, CEOUT => CE_TMP) ;  ← ポートマップ宣言
C2:counter100
    port map(CE => CE_TMP, RESET => RESET, CNT1 => CNT1_TMP, CNT2 => CNT2_TMP) ;
C3:decoder8
    port map(DIN => CNT1_TMP, DOUT => DOUT1) ;
C4:decoder8
    port map(DIN => CNT2_TMP, DOUT => DOUT2) ;
end Behavioral;
```

リスト 5.10 100秒カウンタ階層設計用（watch100）のコード

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity counter100 is
    Port (CE : in std_logic;
          RESET : in std_logic;
          CNT1 : out std_logic_vector(3 downto 0) ;
          CNT2 : out std_logic_vector(3 downto 0));
end counter100;

architecture Behavioral of counter100 is
       signal WORK1 : std_logic_vector(3 downto 0) ;
       signal WORK2 : std_logic_vector(3 downto 0) ;
       signal CE100 : std_logic ;  ← 2桁目の7セグメントLED動作用信号CE100の宣言

begin
process(CE)  ← 1桁目の7セグメントLEDの動作
begin
       if (CE'event and CE='1') then
               if (RESET = '0') then
                        WORK1 <= "0000" ;
               elsif(WORK1 = "1001") then
                        WORK1 <= "0000" ;
               else
                        WORK1 <= WORK1 + '1' ;
       end if;
   end if;
end process;

process(WORK1)  ← 2桁目の7セグメントLED動作用信号CE100の生成
begin
       if (WORK1 = "1001") then
              CE100 <= '1' ;
       else
              CE100 <= '0' ;
       end if;
```

第5章

```
end process;

process(CE)
begin
        if (CE'event and CE='1') then
                if (RESET = '0') then
                        WORK2 <= "0000" ;
                elsif (CE100='1') then
                        if(WORK2 = "1001") then
                                WORK2 <= "0000" ;
                        else
                                WORK2 <= WORK2 + '1' ;
                        end if ;
                end if;
        end if;
end process;

CNT1 <= WORK1 ;
CNT2 <= WORK2 ;

end Behavioral ;
```

2桁目の7セグメントLEDの動作

各7セグメントLEDへの出力代入処理

リスト5.11　100秒カウンタ（counter100）のコード

リスト5.11では，2桁目の7セグメントLEDを動作させる信号*CE*100を生成するprocess文を記述しています．このprocess文によって，1桁目の7セグメントLEDの表示が「9」のときに，*CE*100は「1」に立ち上ります．2桁目の7セグメントLEDは，分周後のクロック*CE*に同期しながら*CE*100の値によってカウントを進めます．

表5.4に，HDLトレーナーのピン割り当てを示します．プッシュスイッチSW1を押すと同期リセットが働くことなどを確認してください．

表5.4　例題5-2のピン割り当て

記号		ピン番号	備考	
入力	*CLK*	22	CLK 1	
	RESET	34	SW1	
出力	*DOUT1*(0)	50	LED4（LEDD）	dp
	DOUT1(1)	61		g
	DOUT1(2)	60		f
	DOUT1(3)	58		e
	DOUT1(4)	56		d
	DOUT1(5)	55		c
	DOUT1(6)	53		b
	DOUT1(7)	52		a
	DOUT2(0)	64	LED3（LEDC）	dp
	DOUT2(1)	74		g
	DOUT2(2)	73		f
	DOUT2(3)	72		e
	DOUT2(4)	71		d
	DOUT2(5)	70		c
	DOUT2(6)	68		b
	DOUT2(7)	67		a

▶ 演習問題 5

5.1　図 5.28 は，2 個の半加算器を用いた全加算器の構成図である．この回路を VHDL によって階層設計したリスト 5.12 の①〜⑮に適当な語句を入れてコードを完成させなさい．

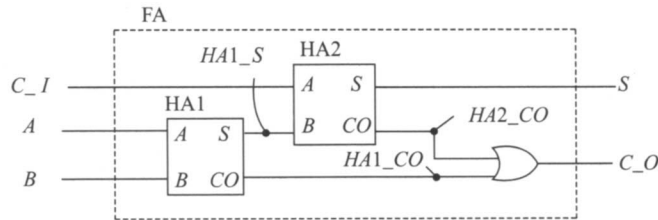



図 5.28　半加算器による全加算器の構成

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

① fa is
    Port ( C_I : in std_logic;
           A : in std_logic;
           B : in std_logic;
           S : out std_logic;
           C_O : out std_logic);
end fa;

architecture Behavioral of fa is
② ha
    Port ( A : in std_logic;
           B : in std_logic;
           S : out std_logic;
           CO : out std_logic);
③ ;
    signal HA1_S : std_logic ;
    signal HA1_CO : std_logic ;
    signal ④ : std_logic ;
begin

C1: ⑤
 port map(A => ⑥, B => B, ⑦ => HA1_S, CO => ⑧) ;
C2:ha
 port map(A => ⑨, B => HA1_S, S => S, CO => HA2_CO) ;

C_O <= HA1_CO ⑩   HA2_CO ;

end Behavioral;
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

⑪ ⑫ is
    Port ( A : in std_logic;
           B : in std_logic;
           S : out std_logic;
           CO : out std_logic);
end ⑬ ;

architecture Behavioral of ha is
begin
S <= A ⑭ B ;
CO <= A ⑮ B ;

end Behavioral;
```

リスト 5.12　全加算器のコード

5.2　階層設計を行う利点を挙げなさい．

5.3　コンポーネント宣言とポートマップ宣言の書式について説明しなさい．

第６章
シミュレーションの基礎

VHDLで設計した回路を実機にダウンロードして動作させる前に，ソフトウェア上で回路の動作をシミュレーションすれば，トラブルを事前に発見することが可能になります．設計者の考えた理論どおりに回路が動作しているか否かを，タイムチャートをみながら視覚的に検討できます．また，回路に入力する各種の信号を実際に準備しなくても，シミュレータ上でいろいろな信号を作成して回路に与えることができます．シミュレーション用のソフトウェア（シミュレータ）としては，メンターグラフィック社のModelSimが有名です．このシミュレータは，ザイリンクス社のISE WebPACKに組み込んで使用できる無償版MXE II Starterが配布されています．この章では，MXE II Starterを使用したシミュレーションを実習しましょう．

6.1 テストベンチ

シミュレーションを行うためには，テストベンチ（test bench）について理解しておくことが必要です．テストベンチとは，シミュレーションを行う回路に与える入力信号や，出力信号を観測するために記述するコードのことです．ここでは，テストベンチについての基礎を学びましょう．

▶6.1.1 テストベンチとは

設計した回路が正しく動作するかどうかをシミュレーションでテストする場合には，適当な入力信号を与えて，回路の動作をチェックすることが必要です．たとえば，図6.1に示す半加算器において，図6.2のタイムチャートに示す入力信号A，Bを与えた場合に出力S，Cが正しく変化しているかどうかをチェックします．

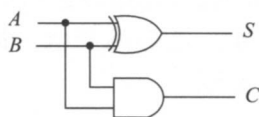



図6.1　半加算器の回路

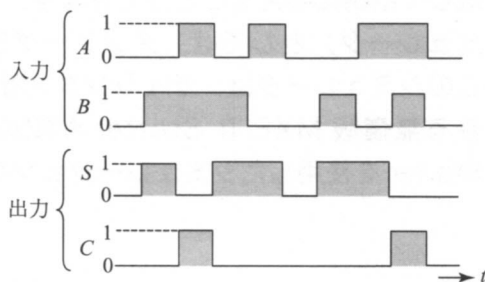



図6.2　半加算器のタイムチャート例

また，シフトレジスタやカウンタなどの同期式の順序回路では，適当なクロック信号を与えて，回路各部の信号の変化を観測します．このように，シミュレーションを行う場合に回路に与える入力信号を記述しておくのがテストベンチです（図6.3参照）．図6.2の例では，入力信号A，Bをテストベンチに記述します．

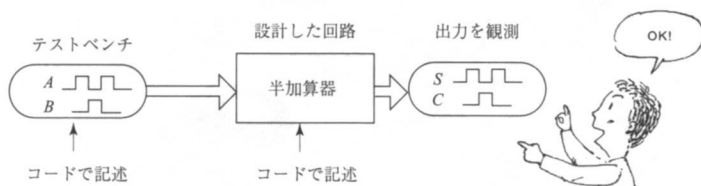



図6.3　テストベンチ

テストベンチに記述した信号は，シミュレーションソフトウェアによって，設計した回路に与えられます．しかし，テストベンチ自体は，ディジタル回路に変換されることはありません．たとえば，テストベンチに方形波を発生するクロック信号を記述したからといって，発振回路が合成されるわけではないのです．

6.1.2 テストベンチの書き方

ここでは，図 6.4 に示す半減算器を例にしてテストベンチの書き方を学びましょう．半減算器を VHDL で記述したコードをリスト 6.1 に示します．

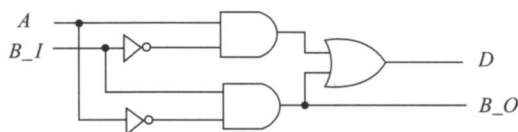



図 6.4 半減算器の回路

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity sample1 is
    Port ( A : in std_logic;
           B_I : in std_logic;
           D : out std_logic;
           B_O : out std_logic);
end sample1;

architecture Behavioral of sample1 is

        signal X, Y:std_logic ;

begin
        X <= A and (not B_I) ;
        Y <= B_I and (not A) ;
        D <= X or Y ;
        B_O <= Y ;
end Behavioral;
```

リスト 6.1 半減算器のコード

リスト 6.2 に，この半減算器用テストベンチの例を示します．与える入力波形の記述方法は，後で詳しく説明します．ここでは，テストベンチ記述のパターンを理解してください．

リスト 6.2 は，設計した回路（半減算器）のコードと区別する意味でアルファベットの大文字を多く使用して記述しましたが，これまでと同様に小文字を中心とした記述でももちろんかまいません．テストベンチでは，コンポーネント宣言とポートマップ宣言を用いて，設計した回路（半減算器）のエンティティ宣言部とインタフェースを取ります．つまり，テストベンチは，設計した回路の上位階層に位置すると考えることができます．

このテストベンチによって，半減算器に与えられる信号 A，B_I の波形パターンを図 6.5 に示します．

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY sample1_tb_vhd_tb IS
END sample1_tb_vhd_tb;

ARCHITECTURE behavior OF sample1_tb_vhd_tb IS

      COMPONENT sample1
      PORT(
           A : IN std_logic;
           B_I : IN std_logic;
           D : OUT std_logic;
           B_O : OUT std_logic
           );
      END COMPONENT;

      SIGNAL A :  std_logic;
      SIGNAL B_I :  std_logic;
      SIGNAL D :  std_logic;
      SIGNAL B_O :  std_logic;

BEGIN

     uut: sample1 PORT MAP(
              A => A,
              B_I => B_I,
              D => D,
              B_O => B_O
     );

   tb : PROCESS
   BEGIN
        A <= '0' ; B_I <= '0' ;
        WAIT FOR 20 ns;          A <= '1' ;
        WAIT FOR 10 ns;          B_I <= '1' ;
        WAIT FOR 10 ns;          A <= '0' ;
        WAIT FOR 10 ns;          B_I <= '0' ;

   END PROCESS;

END;
```

コンポーネント宣言（COMPONENT sample1 を指す）

半減算器の信号宣言．in や out などのモードは不要（SIGNAL A を指す）

ポートマップ宣言（uut: sample1 PORT MAP(を指す）

このテストベンチ例では，process 文中に記述（tb : PROCESS を指す）

半減算器に与える入力信号のパターンを記述（A <= '0' ; B_I <= '0' ; を指す）

リスト 6.2　半減算器のテストベンチ例

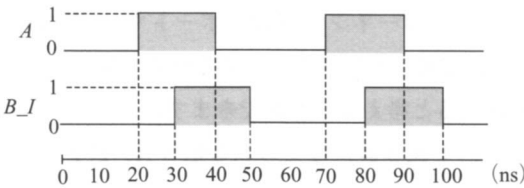



図 6.5　半減算器に与えられる波形パターン

第6章

与える波形パターンの代表的な記述法は，以下の三つです．

① 1/2 周期で状態が変化する繰り返し波形

一般的な方形波のクロック信号のように，'0'と'1'の時間が同じ周期で繰り返される波形です（図 6.6 参照）．初期値の設定は，signal 文では省略して，アーキテクチャ宣言内の機能宣言部で記述することもできます．ただし，signal 文で初期値を設定する場合には，信号代入文（<=）ではなく変数代入文（:=）を用いることに注意してください．

(a) 書式

(b) 記述例

```
signal   CLK  : std_logic : ='0' ;
                  ≀
CLK <= not  CLK  after  50 ns  ;
```

(c) 波形パターン

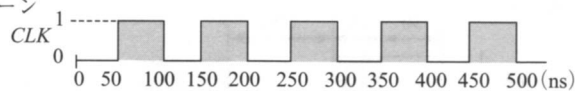



図 6.6 1/2 周期で変化する繰り返し波形

② 任意の時間で変化する繰り返し波形

任意の時間で変化する繰り返し波形を記述する方法です．波形変化の指定は process 文中に記述します（図 6.7 参照）．

(a) 書式

```
signal 信号名 : データ型 := ' 初期値 '
                ≀
process
begin
   wait for 時間 ; 信号名 <= ' 値 ' ;
                  ≀
end process ;
```

(b) 記述例

```
signal CLK : std_logic := '0'
                  ≀
process
begin
   wait for 20ns ; CLK <= '1' ;
   wait for 30ns ; CLK <= '0' ;
end  process;
```

(c) 波形パターン

CLK 1 / 0 — 20 50 70 100 120 150 (ns)

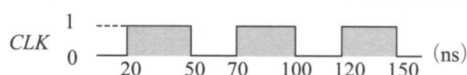



図 6.7 任意の時間で変化する繰り返し波形

このときのprocess文には，センシティビティリストが不要です．また，記述する時間は，初期状態からの相対的な値を示します．初期値の設定は，signal文では省略してprocess文中で記述することもできます（リスト6.2参照）．

③ 任意の時間で変化する有限時間の波形

任意の時間で変化する有限時間の波形を記述する方法です．記述する時間は，初期状態からの絶対的な値を示します（図6.8参照）．

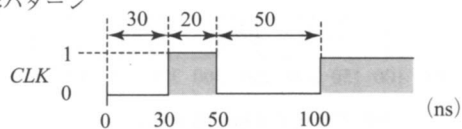



図6.8　任意の時間で変化する有限時間の波形

この他，VHDLの繰り返し制御用の「for文」を使用して波形パターンを記述することも可能です．また，ISE WebPACKには，別ファイルに記述した波形パターンを読み込んで使用する機能（TEXTIO）や，図形として入力した波形パターンをテストベンチコードに変換する機能（HDL Bencher）も備わっています．

6.2 シミュレーション実習

論理シミュレーションの実習を行って，設計した回路の動作を検証しましょう．ここでは，MXE II Starterがインストールされていることを前提にして実習を進めます．インストール法などについては，第7章を参照してください．

▶6.2.1 シミュレーションの手順

リスト6.1（151ページ）で扱った半減算器のシミュレーションを例にして，実際の操作手順を説明します．

① 半減算器のコード記述

ここでは，プロジェクト名を「sample1」としました．記述を終えたら「Sysnthesize-XST」を実行してエラーのないことを確認しておきましょう（図6.9参照）．

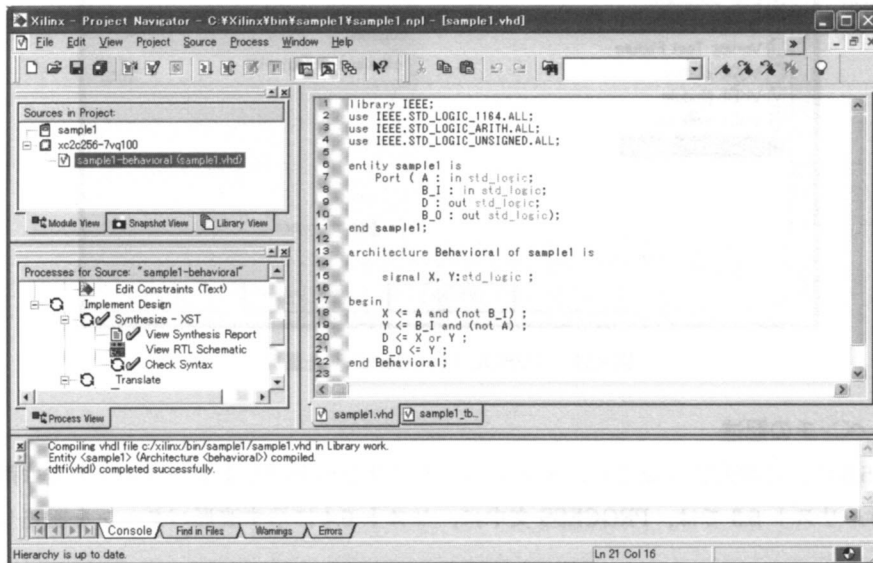



図6.9　半減算器のVHDLコードを記述

② テストベンチ記述用ウインドウの準備

ツールバーの「Project」→「New Source」を選択し（図6.10），図6.11に示す「New Source」ウインドウを表示します．このウインドウで，ファイル名をたとえば「tb」として，ウインドウ

左側で「VHDL Test Bench」を選択し「次へ」ボタンをクリックします．続いて現れるいくつかのウインドウは，そのまま進めていきます．

すると，テストベンチを記述するひな形の入ったウインドウが表示されます．

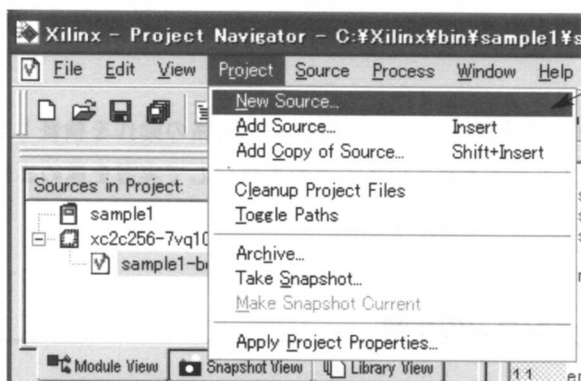



図6.10　「Project」→「New Source」を選択

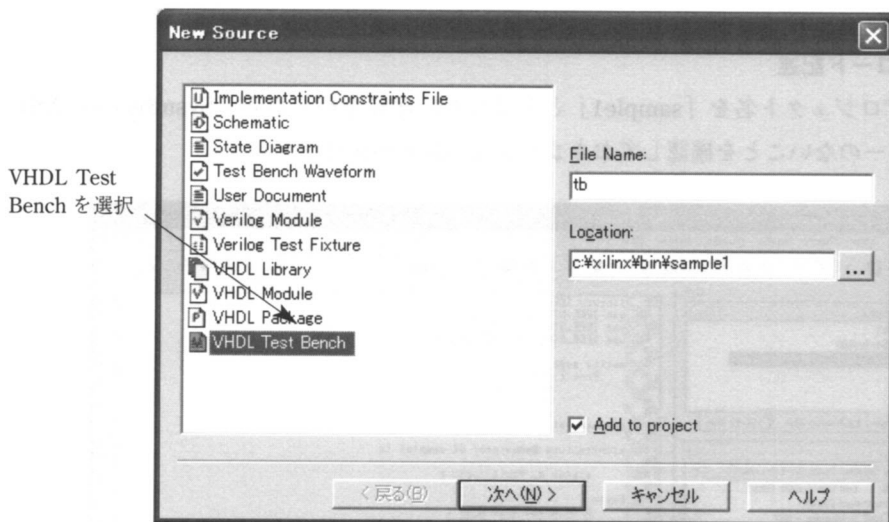



図6.11　「VHDL Test Bench」を選択

③ テストベンチの記述

リスト6.3に，自動的に作成されたテストベンチのひな形を示します（コメント文は削除してあります）．リスト6.3では，PROCESS文中に，リスト6.4に示す波形パターンを記述しましょう．

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY sample1_sample1_tb_vhd_tb IS
END sample1_sample1_tb_vhd_tb;

ARCHITECTURE behavior OF sample1_sample1_tb_vhd_tb IS

       COMPONENT sample1
       PORT(
              A : IN std_logic;
              B_I : IN std_logic;
              D : OUT std_logic;
              B_O : OUT std_logic
              );
       END COMPONENT;

       SIGNAL A :  std_logic;
       SIGNAL B_I :  std_logic;
       SIGNAL D :  std_logic;
       SIGNAL B_O :  std_logic;

BEGIN

     uut: sample1 PORT MAP(
            A => A,
            B_I => B_I,
            D => D,
            B_O => B_O
     );

   tb : PROCESS
   BEGIN


   END PROCESS;


END;
```

この部分に，波形パターンを記述する

リスト 6.3　自動的に作成されたテストベンチのひな形

```
A <= '0' ; B_I <= '0' ;
WAIT FOR 20 ns ;    A <= '1' ;
WAIT FOR 10 ns ;    B_I <= '1' ;
WAIT FOR 10 ns ;    A <= '0' ;
WAIT FOR 10 ns ;    B_I <= '0' ;
```

リスト 6.4　波形パターン

④ シミュレータの起動

「Module View」ウインドウでテストベンチファイルを選択した後，「Process View」ウインドウの「ModelSim Simulator」の下位リスト部分にある「Simulator Behavioral Model」をダブルクリックして，MXE II Starter を起動します（図 6.12 参照）．

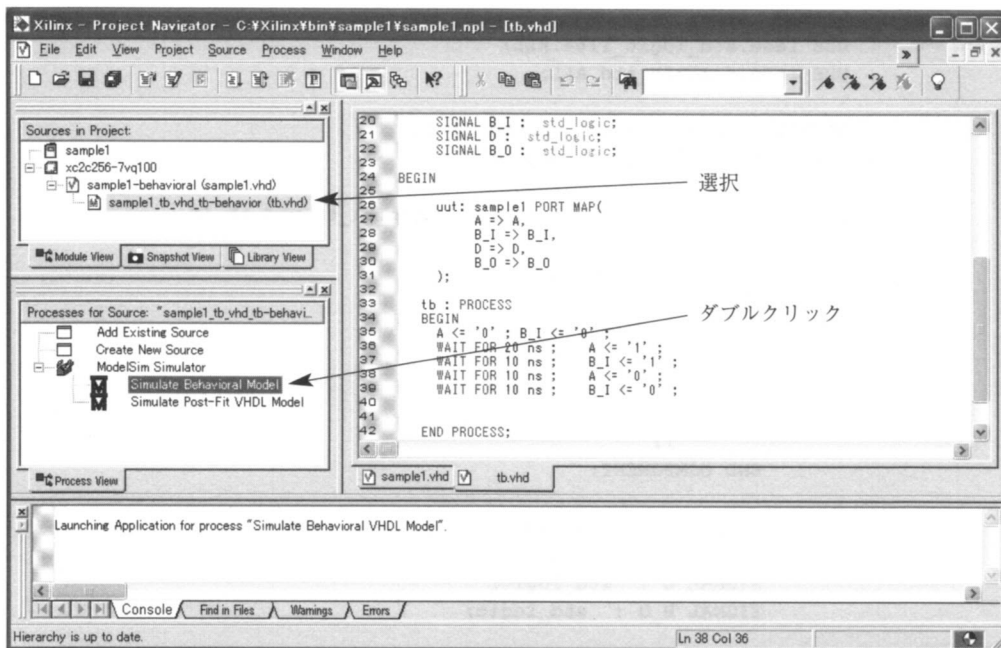



図 6.12　MXE II Starter の起動

⑤ **シミュレータのリセット**

シミュレータが起動すると，図 6.13 に示すように「メインウインドウ」，「シグナルウインドウ」，「ストラクチャウインドウ」，「ウェーブウインドウ」が表示されます．

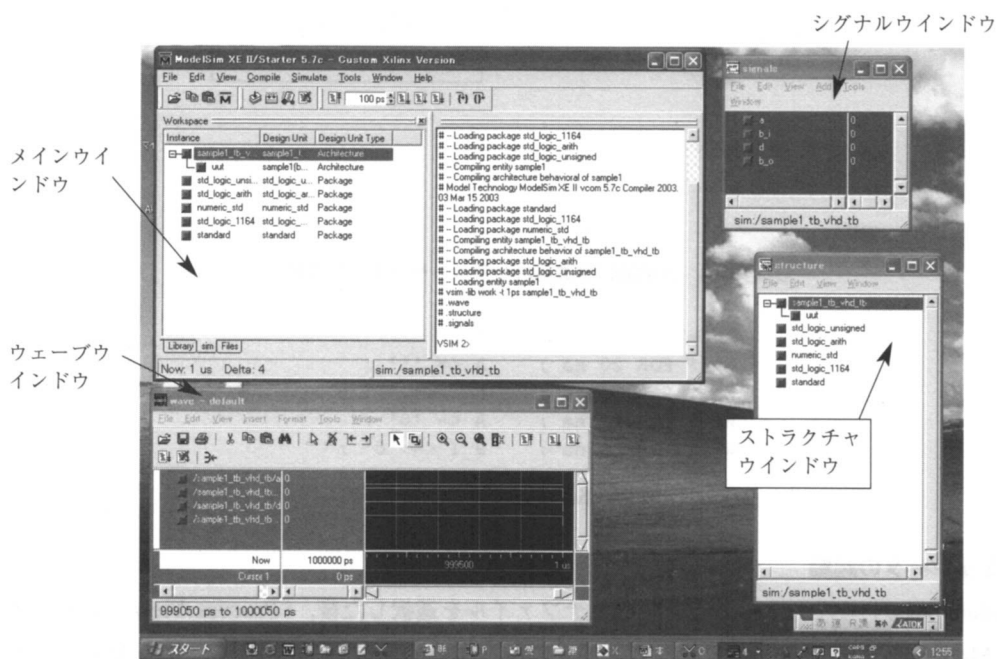



図 6.13　シミュレータの起動画面

図 6.14 に示す「ウェーブウインドウ」の「リスタート」ボタンをクリックして現れるウインドウ（図 6.15 参照）で「Restart」をクリックして，「ウェーブウインドウ」をリセットします．

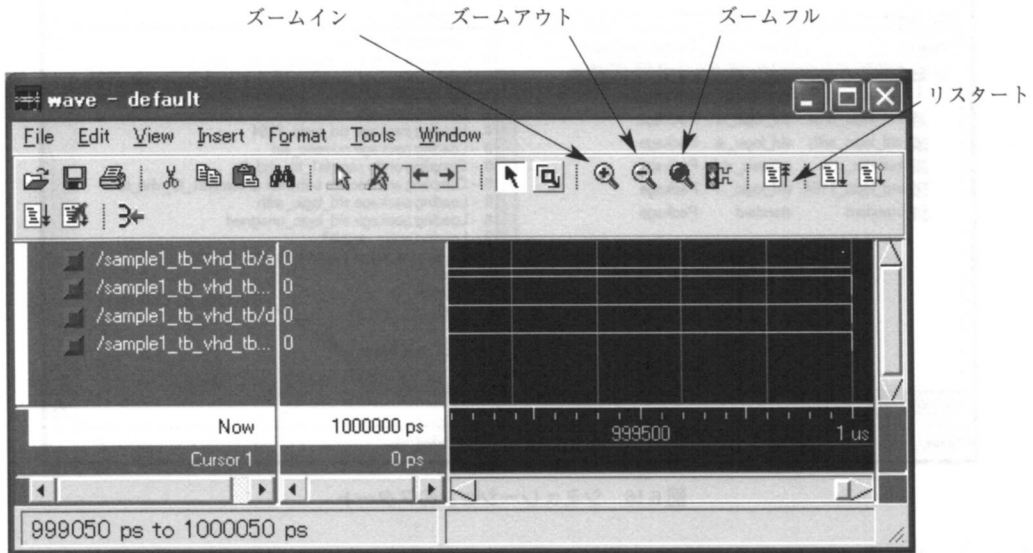



図 6.14　「ウェーブウインドウ」

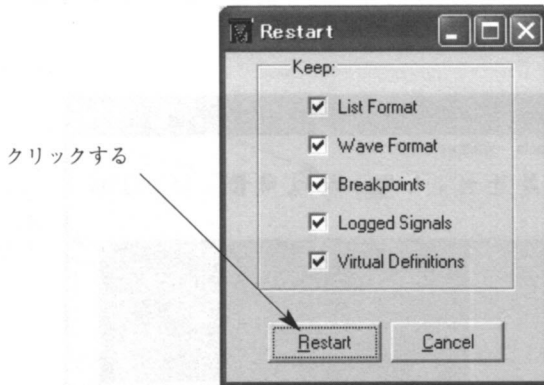



図 6.15　リスタートウインドウ

⑥ シミュレーションのスタート

「メインウインドウ」から，シミュレーションを行う時間を入力してシミュレーションをスタートさせます．たとえば，100 ns のシミュレーションを行う場合には，「run 100 ns」と入力します（図 6.16 参照）．

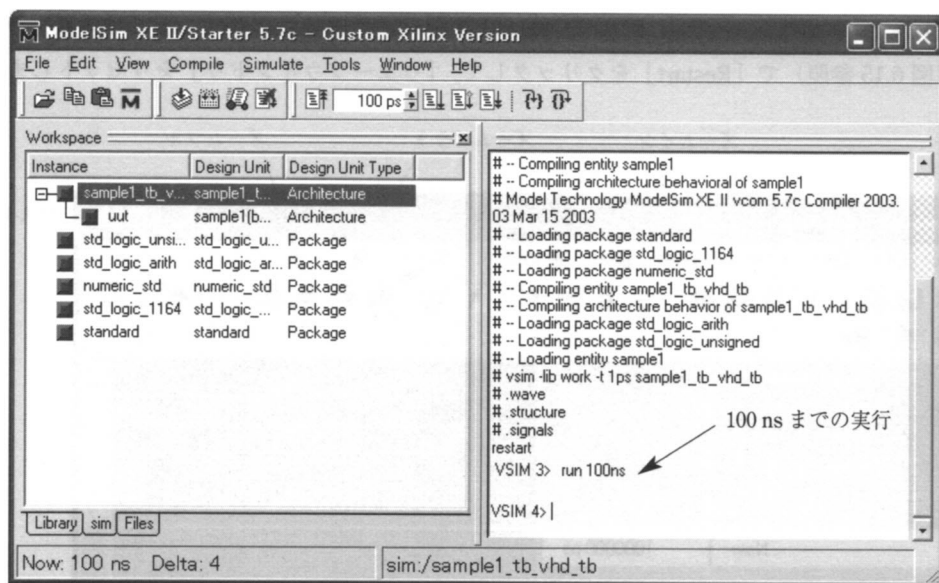



図6.16　シミュレーションのスタート

⑦ シミュレーション結果の表示

「ウェーブウインドウ」の「ズームフル」ボタンをクリックした後，「ズームイン」や「ズームアウト」ボタンをクリックして見やすい表示になるように調整します（図6.17参照）．

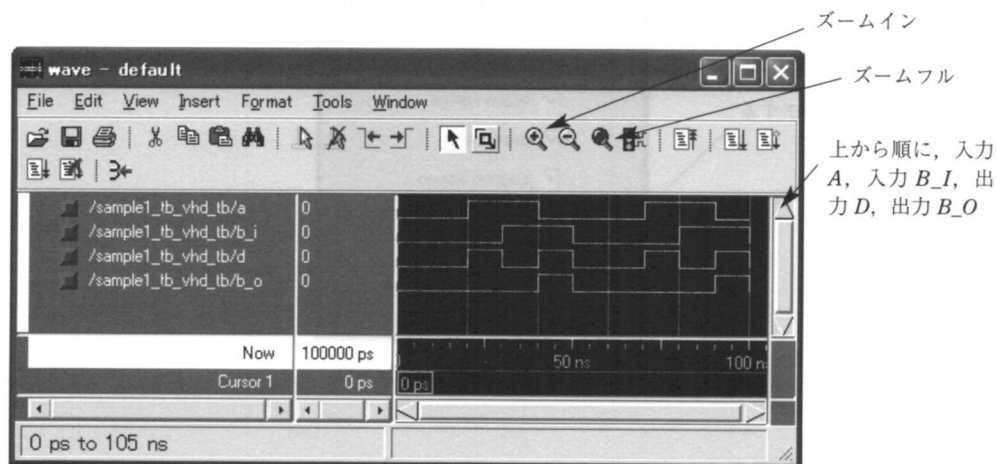



図6.17　半減算器のシミュレーション画面

⑧ タイムチャートの検討

シミュレーションによって得られたタイムチャートを検討して，設計した回路（半減算器）の動作を検討しましょう．

MXE II Starterを終了する場合には，メインウインドウの右上の ☒ をクリックします．

▶6.2.2　順序回路のシミュレーション

シミュレーション実習として，同期式カウンタの動作を確認してみましょう．

例題 6-1

同期式 10 進カウンタ（121 ページ例題 4-10）の動作をシミュレーションによって確認しなさい．ただし，同期リセットとすること．

解答例

リスト 6.5 に 10 進カウンタのコードを示します．

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity rei4_10 is
    Port ( CLK : in std_logic;
           RESET : in std_logic;
           Q : out std_logic_vector(3 downto 0));
end rei4_10;

architecture Behavioral of rei4_10 is
       signal WORK : std_logic_vector(3 downto 0) ;
begin
process(CLK)
begin
       if (CLK'event and CLK='1') then
              if (RESET = '0') then
                     WORK <= "0000" ;
              elsif(WORK = "1001") then
                     WORK <= "0000";
              else
                     WORK <= WORK + '1' ;
            end if;
       end if;
end process;
Q <= WORK ;
end Behavioral;
```

リスト 6.5　10 進カウンタのコード

このカウンタの入力信号は，クロック *CLK* とリセット *RESET* です．これらの波形パターンをたとえば，図 6.18 のように設定します．

つまり，*CLK* は 20 ns ごとに変化する 25 kHz の波形，*RESET* は 500 ns ごとに変化する 1 kHz の波形とします．どちらも繰り返し波形であるとすると，153 ページの① 1/2 周期で状態が変化する繰り返し波形の書式で記述できます．リスト 6.6 にテストベンチを示します．

メインウインドウで「run 2000 us」（us は μs を意味します）と入力してシミュレーションを行った画面を図 6.19 に示します．*RESET* が，'1'（時間 500 μs）になるとカウンタが動作し，*RESET* が '0'（時間 1 ms）になると *CLK* に同期してリセットがかかっている様子が確認できます．また，図 6.20 に示すように出力 *Q* の部分を右

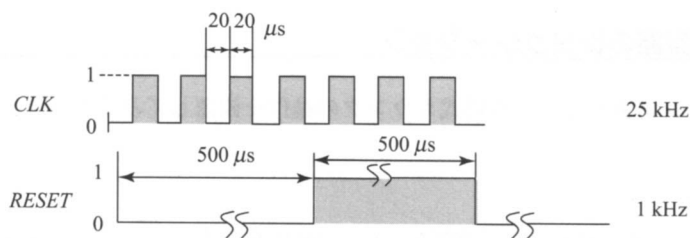



図 6.18　波形パターンの設定

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY rei6_1_tb_vhd_tb IS
END rei6_1_tb_vhd_tb;

ARCHITECTURE behavior OF rei6_1_tb_vhd_tb IS

        COMPONENT rei6_1
        PORT(
                CLK : IN std_logic;
                RESET : IN std_logic;
                Q : OUT std_logic_vector(3 downto 0)
                );
        END COMPONENT;

        SIGNAL CLK : std_logic:='0';
        SIGNAL RESET : std_logic:='0';
        SIGNAL Q :  std_logic_vector(3 downto 0);

BEGIN

        uut: rei6_1 PORT MAP(
                CLK => CLK,
                RESET => RESET,
                Q => Q
        );

        CLK <= not CLK after 20 us;
        RESET   <= not RESET after 500 us;

END;
```

リスト 6.6　テストベンチ

クリックして，たとえば「Radix（基数）」を変更すれば，表示形式を変えることができます．

波形データは，「ウェーブウインドウ」のツールバーの「File」→「Save Dataset」→「sim」を選択して保存することができます．保存した波形データの読み込みは，「メインウインドウ」のツールバーの「File」→「Open」→「Dataset」で読み込むファイル名を指定します．その後，図 6.21 に示すように操作すれば「ウェーブウインドウ」が表示されます．

図 6.19 2000 μs（2 ms）までのシミュレーション結果の一部

10 進カウンタの動作が容易に見てとれる

図 6.20 出力 Q の表示形式の変更

右クリック

選択する

図 6.21 保存した波形データの読み込み操作

▶ 演習問題 6

6.1 つぎの①から⑤の記述で正しいものはどれか答えなさい．

① VHDLでは，発振回路を記述して構成することができる．
② テストベンチで記述したコードは，回路に合成されない．
③ 論理シミュレーションでは，回路の遅延を考慮しない．
④ 論理シミュレーションは，必ず行わなければならない．
⑤ テストベンチは，テストする回路よりも下位の階層に位置する．

6.2 リスト6.7，リスト6.8にテストベンチのコードの一部を示す．これにより，生成される各波形パターンを図示しなさい．

```
signal CLK : std_logic:= '1'
CLK <= not  CLK after 80 ns
```

リスト6.7

```
CLK <= '1' , '0' after 20 us
             '1' after 50 us
             '0' after 100 us
```

リスト6.8

6.3 図6.22に示す波形パターンは，80 nsまでのパターンを繰り返すものである．この波形パターンを生成するテストベンチの一部であるリスト6.9の①から⑤に適当な語句を入れなさい．

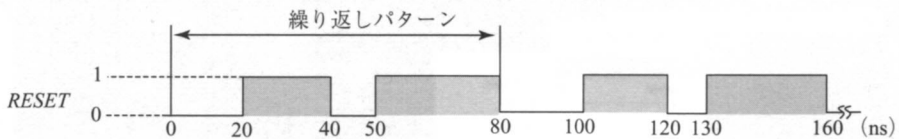



図6.22　波形パターン

```
signal RESET : std_logic : =① ;
②
begin
        wait for 20ns ; RESET <= ③ ;
        wait for 20ns ; RESET <= '0' ;
        wait for ④ns ; RESET <= '1' ;
        wait for 30ns ; RESET <= '0' ;
⑤ ;
```

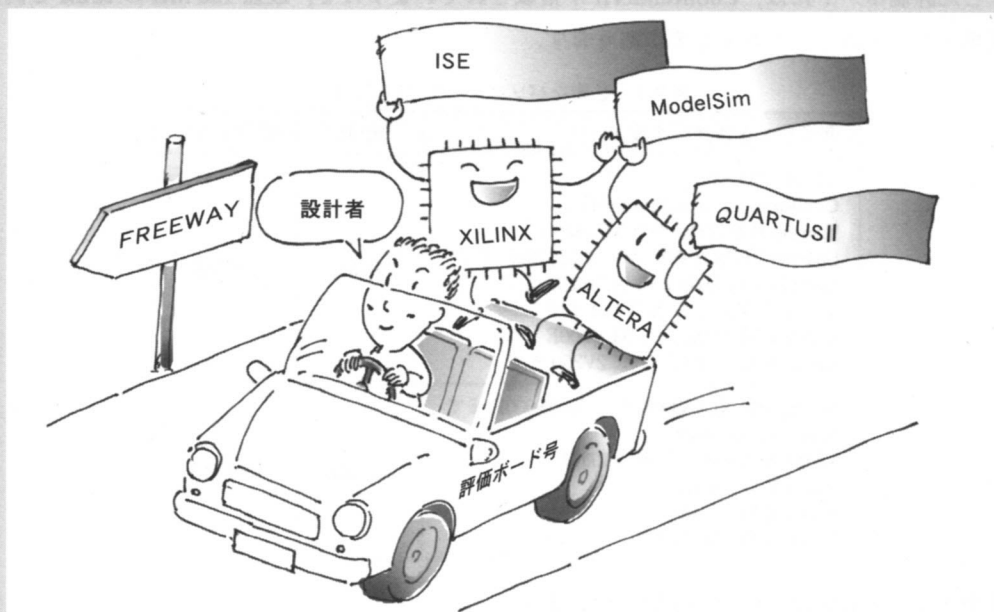
リスト6.9　テストベンチの一部

6.4 VHDLで記述した半加算器の回路をシミュレーションによって動作確認しなさい．

6.5 VHDLで記述した同期式5進カウンタの回路をシミュレーションによって動作確認しなさい．

第6章

第7章
開発ツール

HDLを用いた回路設計に使用する開発ツールは，CPLD/FPGAメーカのホームページから無償でダウンロードして使用することができます．フリーとはいえ，多くは非常に高性能なソフトウェアです．この章では，ザイリンクス社のISE WebPACK，およびアルテラ社のQuartusII Web Editionをダウンロードしてインストールする手順を説明します．また，メンターグラフィックス社のシミュレータModelSimの無償版についても同様の説明を行います．そして，HDLをマスターするのには不可欠な評価ボードの回路などについても理解しておきましょう．

7.1 ザイリンクス社の開発ツール

ザイリンクス社の提供している開発ツールは，ISEとよばれています．ISEには，いくつかの種類がありますが，フリーで入手可能なのはISE WebPACKです．本書執筆時（2004.2）のISE WebPACKの最新バージョンは，6.1iです．ここでは，例としてWindows XPにISE WebPACK 6.1iをインストール方法について説明します．

▶7.1.1　ISE WebPACKの動作環境

有償版のISE FoundationなどはWindows 2000/XPに加えてLinuxとSunSolarisにも対応していますが，無償版のISE WebPACK6.1iのサポートするOSはWindows 2000/XPのみです．Windows 2000を使用する場合には，サービスパック2以上も必要となります．

パソコンは，Pentiumクラス500 MHz以上のCPUが推奨されています．パソコンと評価ボードをパラレルインタフェースで接続するためには，パソコンにプリンタポートが備わっていることが必要です．また，パソコンに必要なメモリ容量は，ターゲットとなるCPLD/FPGAの機種によって異なります．表7.1に，必要なパソコンメモリ容量の例を示します．たとえば，本書で紹介した評価ボードには，CoolRunnerIIが搭載されていますので，最低128 MBのRAMと同量の仮想メモリを用意しておく必要があります．

表7.1　必要なパソコンメモリ容量の例

ザイリンクスデバイス	RAM	仮想メモリ
XC9500/XL/XV CoolRunner，CoolRunner-II Spartan-II XC2S15からXC2S200まで Spartan-IIE XC2S50EからXC2S200Eまで Spartan-3 XC3S50からXC3S200まで Virtex XCV50からXCV150まで Virtex-E XCV50EからXCV200Eまで Virtex-II XC2V40からXC2V250まで	128 MB	128 MB
Spartan-IIE XC2S300EからXC2S600Eまで Spartan-3 XC3S400 Virtex XCV300からXCV400まで Virtex-E XCV400E Virtex-II XC2V500 Virtex-II Pro XC2VP2	256 MB	256 MB

仮想メモリの設定は，つぎの手順で行います．

Windows XPの「スタート」ボタン→「コントロールパネル」→（「パフォーマンスとメンテナンス」）→「システム」を選択して「システムのプロパティ」ウインドウを開きます．開いた

ウインドウで，「詳細設定」タブ→パフォーマンスの「設定」→パフォーマンスオプションの「詳細設定」タブと選択していくと，図7.1に示すパフォーマンスオプションのウインドウが開きますので，そこで「変更」ボタンをクリックすれば設定が行えます．

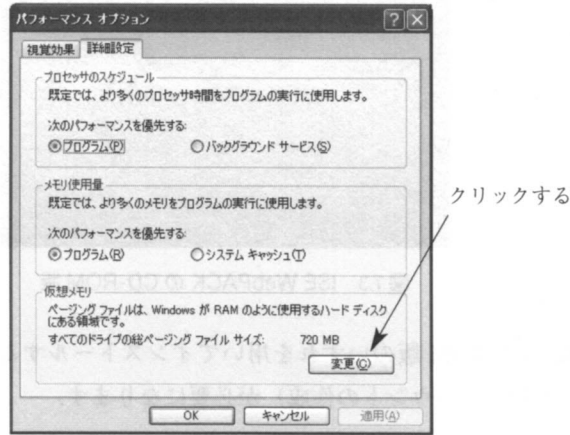



図7.1　仮想メモリの設定（Windows XP）

パソコンには，Internet Explorer 5.0以上，またはNetscape Communicator 4.7以上のブラウザをインストールし，インターネットに接続しておきましょう．

ISE WebPACK 6.1iでは，シミュレータModelSimの無償版であるMEX II Starterを組み込んで統合的な操作を行うことができます（158ページ参照）．

▶7.1.2　ISE WebPACKの入手とインストール

ISE WebPACKは，ザイリンクス社のホームページからダウンロードして入手することができます（図7.2）．また，ザイリンクス社の代理店などからCD-ROM版も配布されています（図7.3）．

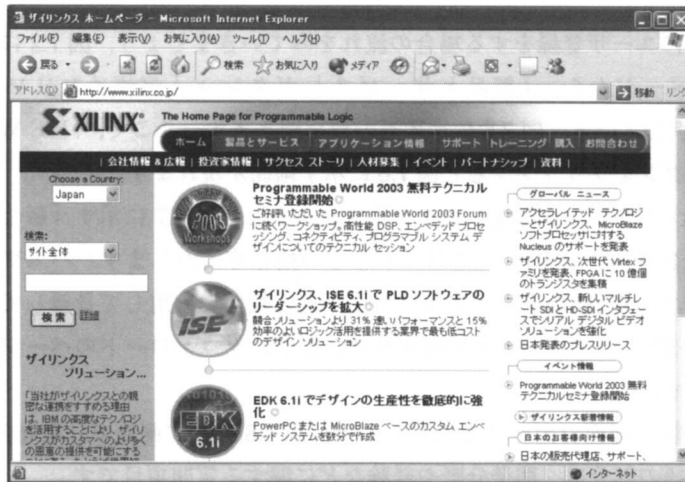



図7.2　ザイリンクス社のホームページ（http://www.xilinx.co.jp/）

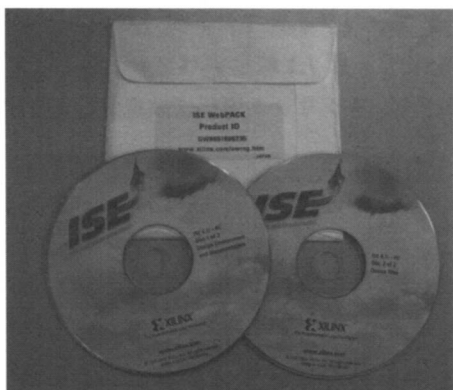



図7.3　ISE WebPACKのCD-ROM版

ダウンロード版，CD-ROM版のいずれを用いてインストールする場合においても，ザイリンクス社へのユーザ登録（アカウントの作成）が必要になります．

① ザイリンクス社のホームページからのダウンロードとインストール法

図7.4に，ISE WebPACK 6.1iをホームページからインストールする場合の流れを示します．

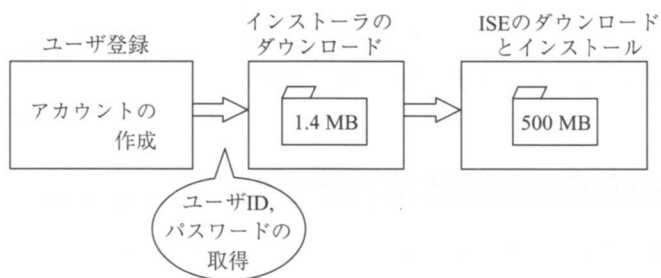



図7.4　ホームページからインストールする流れ（WebInstall）

ここでは，新たにユーザ登録をする場合の流れをみていきましょう．図7.2に示したザイリンクス社のホームページから，「製品とサービス」ボタン→「デザインリソース」→「ISE WebPACK」と選択していき，図7.5に示すページを開きます．ここで，「ISE WebPACKダウンロードの登録」ボタンをクリックすると図7.6に示す画面が現れます．

図7.6の画面で「アカウント作成」ボタンをクリックし，図7.7に示す「新しいアカウントの作成」画面で，氏名やユーザID，パスワードなどの情報を入力します．パスワードは半角の英数字7文字以上，ユーザIDは小文字アルファベットと数字を使用して登録してください．ここで登録したユーザIDとパスワードを使用すれば，以降はダウンロードのページへ直接ジャンプすることができます．

続いて「住所情報」画面でデータを入力した後，図7.8に示す画面で「職務内容などの概要」を入力します．英語の部分は，開発分野や経験などに関する質問項目です．

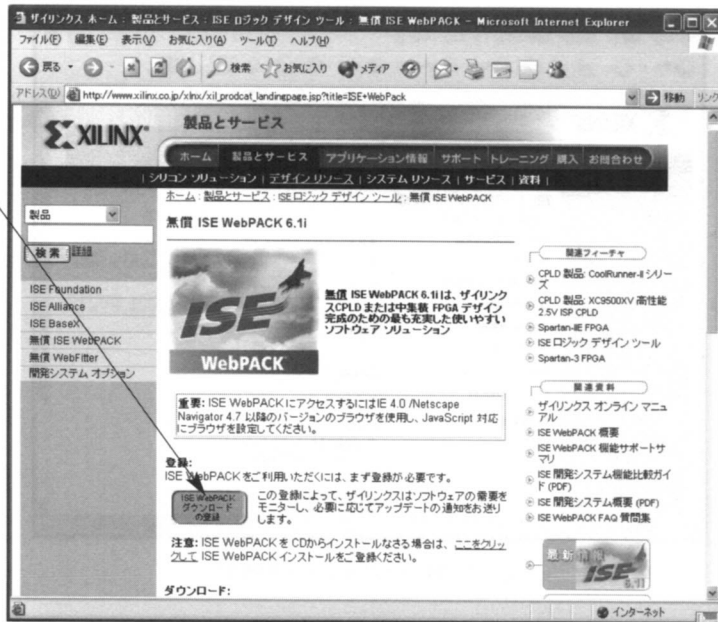



図 7.5 ISE WebPACK ダウンロードのページ

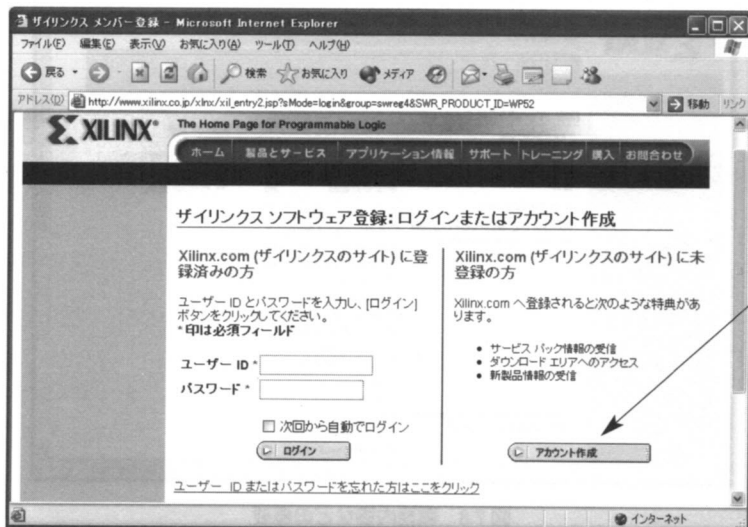



図 7.6 アカウントの作成

入力後，つぎに進み図 7.9 に示すメッセージ画面が現れれば登録は終了です．まもなく登録した電子メールアドレス宛に，ザイリンクス社から 2 通のメールが送られてきます．

これらのメールには，ユーザ ID やパスワード，プロダクト ID などが記されています．また，登録したパスワードや電子メールアドレスなどの変更方法に関する情報も書かれていますので確認しておきましょう．

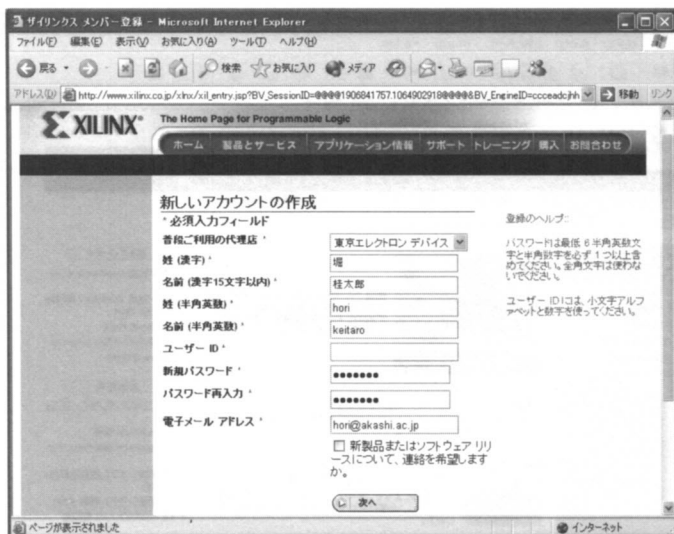



図 7.7　新しいアカウントの作成

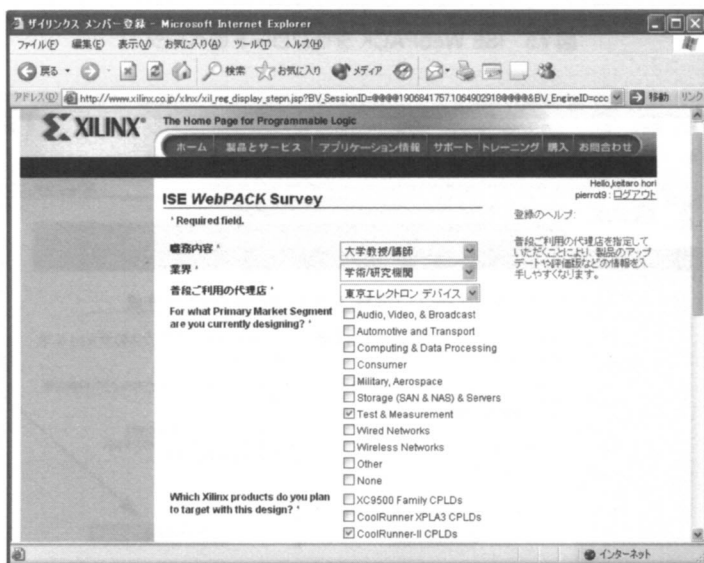



図 7.8　職務内容などの概要

以上の操作で，ISE WebPACK 6.1i をインストールする資格が得られました．図 7.9 の「Download ISE WebPACK」部分をクリックしてダウンロード画面に進みましょう．また，今後は図 7.4 に示した画面からすぐにダウンロード画面に進むこともできます．

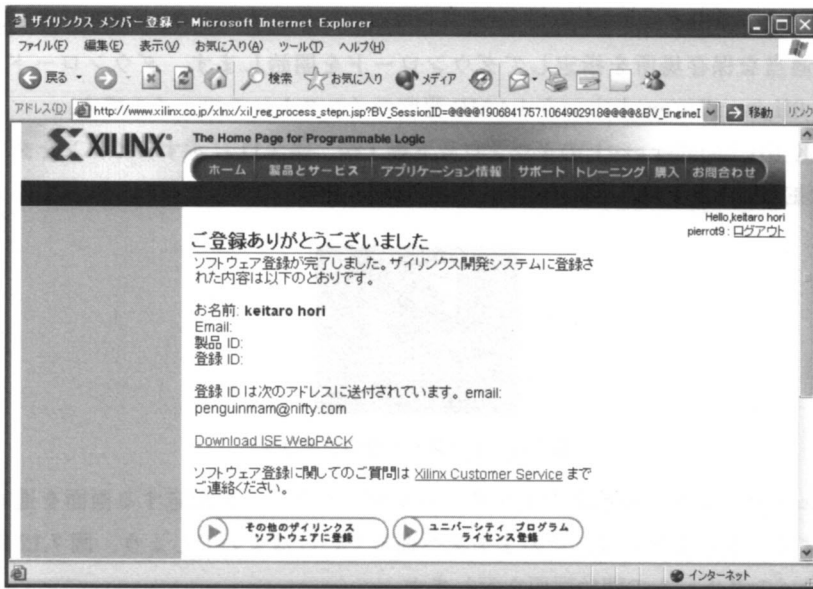



図 7.9　登録終了のメッセージ

ISE WebPACK 6.1i をホームページからインストールする場合には，シングルファイルダウンロード，および WebInstall とよばれる二種類の方法が利用できます．ザイリンクス社では，後者を推奨していますので，図 7.10 の画面において「XILINX WebInstall」の部分をクリックします．

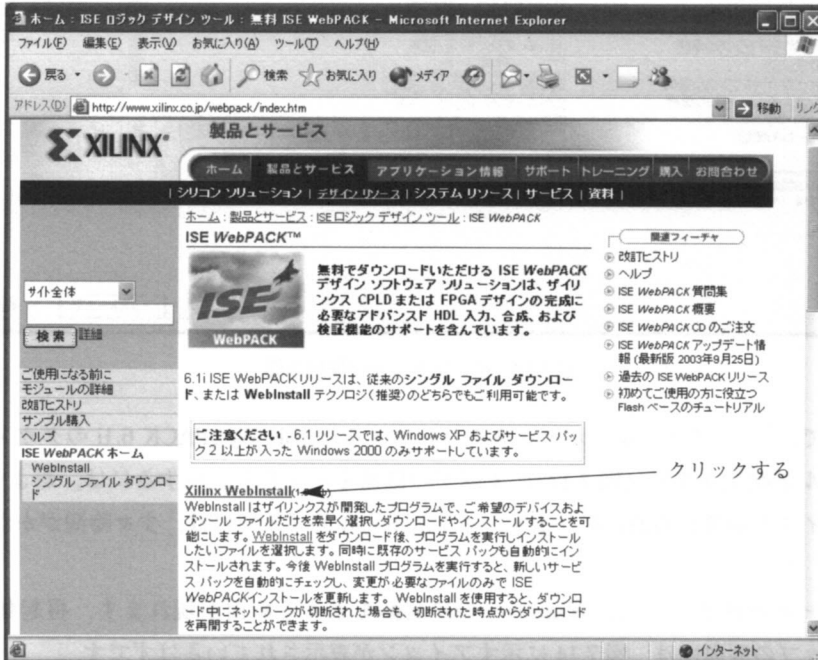



図 7.10　WebInstall を選択

すると，「ファイルのダウンロード」ウインドウが表示されますので，「保存」ボタンをクリックした後，適当な保存場所を指定してダウンロードを開始します．ダウンロードするのは，「WebPACK_61i_installer.exe」という 1.4 MB 程度のインストーラファイルです．

「WebPACK_61i_installer.exe」のダウンロード終了後，図 7.11 に示すアイコンをダブルクリックすれば，ISE WebPACK 6.1i のインストール設定画面が現れます．

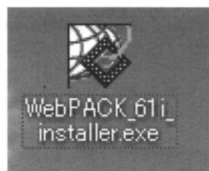



図 7.11　インストーラのアイコン

ソフトウェアライセンスの承諾やインストールディレクトリを指定する画面を進めていきます．内容がよくわからなければ，そのまま次へ進めていけばよいでしょう．図 7.12 に，各種の設定を終えたインストールの開始画面を示します．

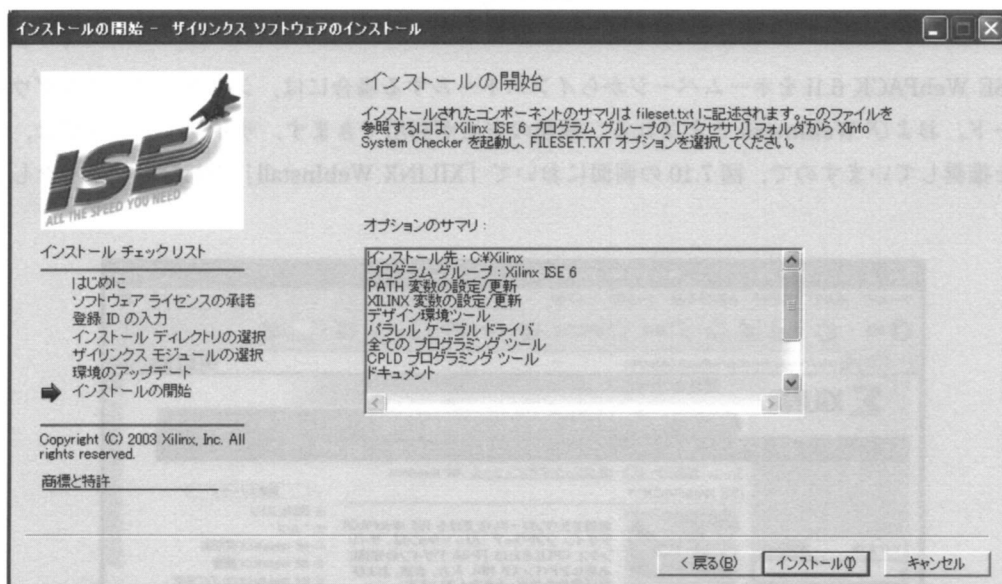



図 7.12　インストールの開始画面

この画面で，「インストール」ボタンをクリックすると ISE WebPACK 6.1i のダウンロードとインストールが始まります（図 7.13）．すべての機能をインストールするには，およそ 500 MB のハードディスク領域が必要になります．インストール終了までには，少々時間がかかることでしょう．

インストールが終了すると，Windows の再起動を求める画面が現れます．再起動を行ったディスクトップの画面には，図 7.14 に示すアイコンが表示されているはずです．

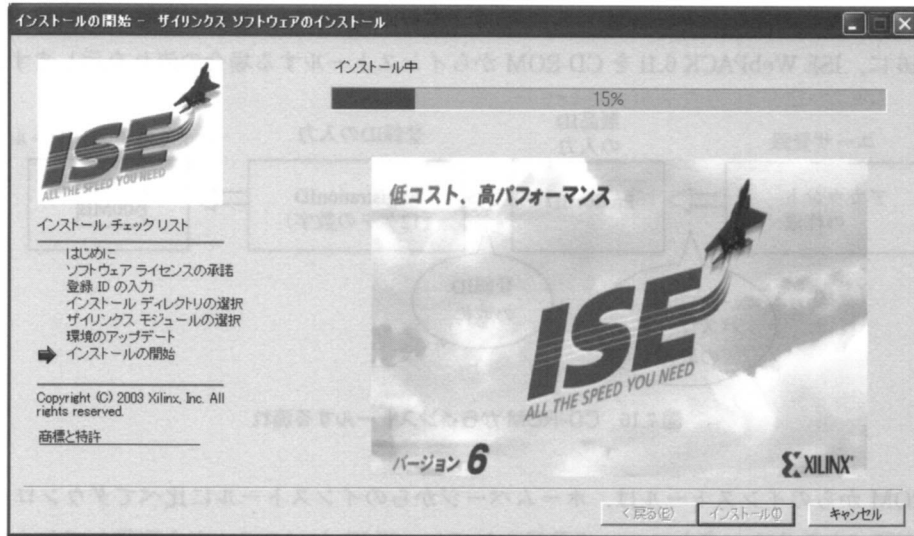



図 7.13 ISE WebPACK 6.1i インストール中

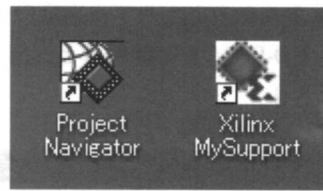



図 7.14 インストール後に表示されるアイコン

以上で，ISE WebPACK 6.1i のインストールは完了です．図 7.14 左側のアイコンをダブルクリックすると ISE WebPACK 6.1i が起動します（図 7.15）．

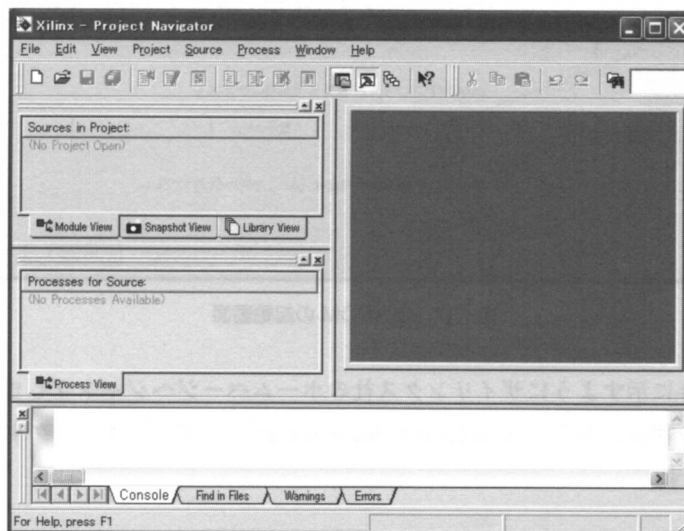



図 7.15 ISE WebPACK 6.1i を起動した画面

第7章

② CD-ROMからのインストール法

図7.16に，ISE WebPACK 6.1iをCD-ROMからインストールする場合の流れを示します．

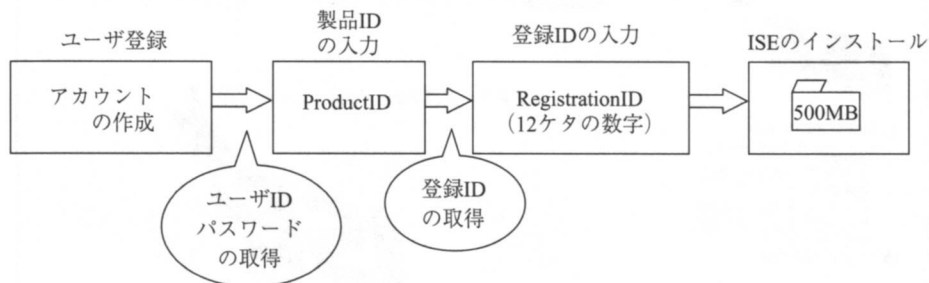



図7.16 CD-ROMからインストールする流れ

CD-ROMからのインストールは，ホームページからのインストールに比べてダウンロードの時間が不要になります．また，ユーザ登録をしてユーザIDとパスワードを取得しておくのに加えて，レジストレーションIDも取得する必要があります．

CD-ROM版のISE WebPACK 6.1iをインストールする実際の手順をみてみましょう．

始めに，ユーザ登録を行います．CD-ROMを起動すると，図7.17に示す画面が現れますので，「Webサイト」ボタンをクリックします．

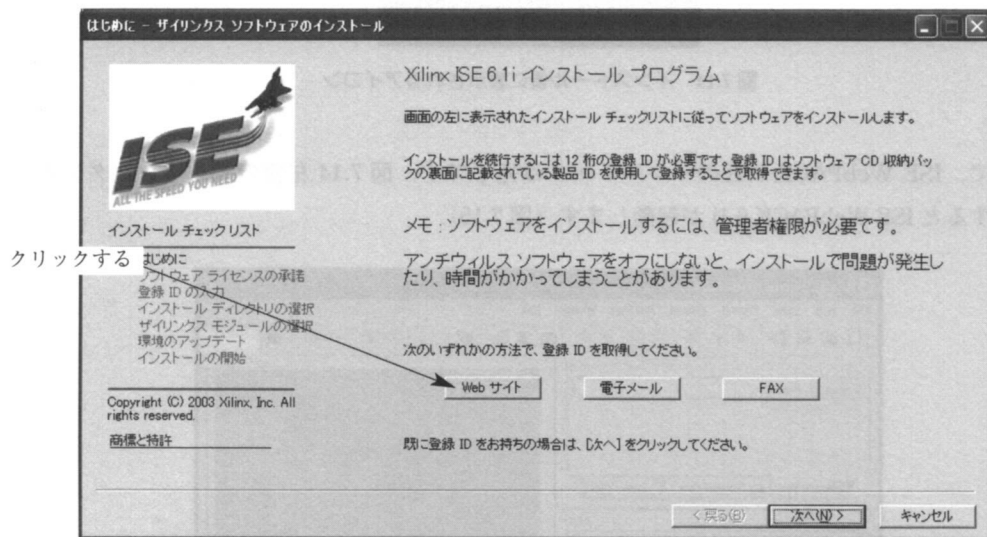



図7.17 CD-ROMの起動画面

すると，図7.18に示すようにザイリンクス社のホームページへジャンプします．ホームページ中の，「ログイン画面に進む」(Continue to login screen)の部分をクリックすると，図7.19に示す画面が現れます．

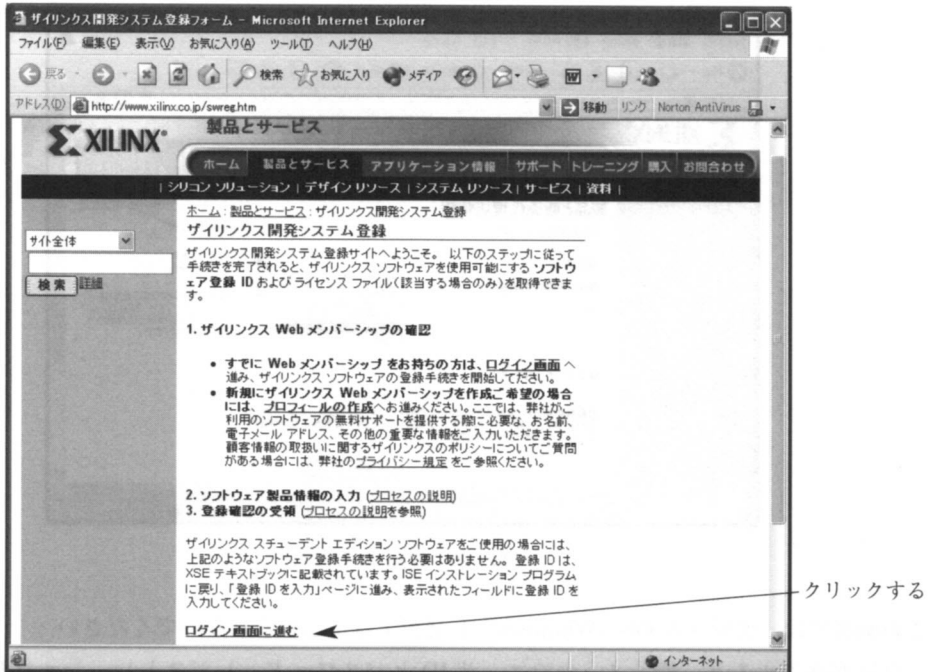



図 7.18 ザイリンクス社のホームページ

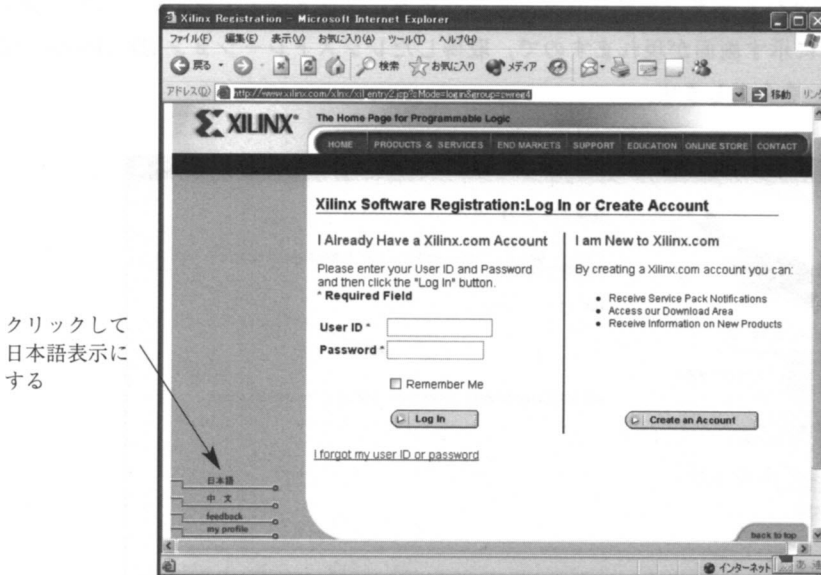



図 7.19 アカウントの作成（図 7.6 と同様）

図 7.19 は，図 7.6 と同じ「アカウントの作成」画面です．もしも英語表示になっている場合は，必要に応じて日本語表示に切り替えるとよいでしょう．ここからは，図 7.6 ～図 7.8 までと同様に氏名や住所，職務内容についての入力を行います．やがて，図 7.20 に示す製品 ID を入力する画面が現れますので，CD-ROM の入っていた袋などに記載されている Product ID を入力します．

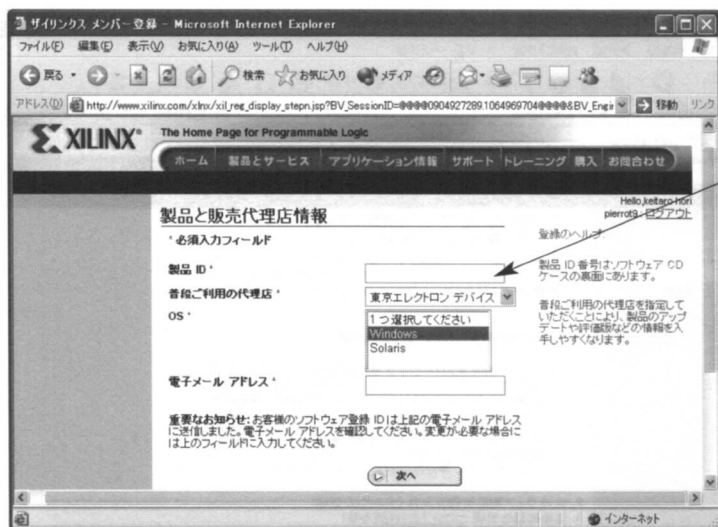



図 7.20　製品 ID の入力

この画面では，使用する OS（Windows）を選択することも忘れないでください．

登録が終了すると，電子メールでユーザ ID とパスワード，レジストレーション ID などが送信されてきますので確認しておきましょう．

図 7.17 へ戻り，「次へ」ボタンをクリックして先へ進み，ソフトウェアライセンスの承諾などを行うと，図 7.21 に示す画面が現れますので，取得したレジストレーション ID（Registration ID : 12 桁の英数字）を入力します．

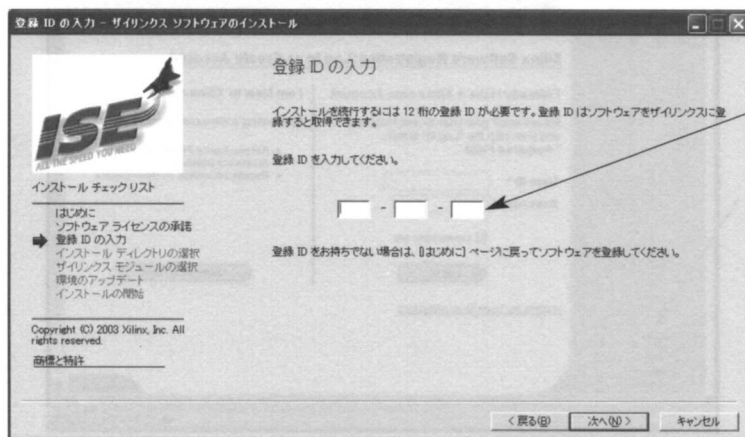



図 7.21　レジストレーション ID の入力

続いて，インストールディレクトリを指定する画面などが現れますので，そのままつぎへ進めばインストールが開始されます．1 枚目に引き続いて 2 枚目の CD-ROM で各種デバイスファイルのインストール を行います．

7.2 アルテラ社の開発ツール

アルテラ社の提供している開発ツールには，Quartus II や MAX+PLUS II があります．ここでは，フリーで入手可能な Quartus II Web Edition を Windows XP パソコンにインストールする方法について説明します．本書執筆時点（2004.2）での，Quartus II Web Edition の最新バージョンは，3.0 です．

7.2.1 Quartus II Web Edition の動作環境

Quartus II Web Edition 3.0 を動作させるためには，Pentium II（400 MHz 以上），メインメモリ 256 MB（512 MB 以上を推奨）を搭載したパソコンに，Windows NT 4.0（サービスパック 3 以上），または Windows 2000/XP がインストールされていることが必要です．また，パソコンを評価ボードとパラレルケーブルで接続するためには，パソコンにプリンタポートが備わっていることが必要です．パソコンには，Internet Explorer 5.0 以上のブラウザをインストールし，インターネットに接続しておきましょう．

表 7.2 に，Quartus II Web Edition 3.0 のサポートするデバイスを示します．

表 7.2　Quartus II Web Edition 3.0 サポートデバイス

デバイス・ファミリ	デバイス	デバイス・ファミリ	デバイス
Cyclone™	全デバイス	FLEX 10K® FLEX 10KA	全デバイス
Stratix™	EP1S10	FLEX® 6000	全デバイス
APEX™ II	EP2A15	MAX® 7000AE	全デバイス
Excalibur™	EPXA1	MAX 7000B	全デバイス
APEX 20KE	EP20K30E EP20K60E EP20K100E EP20K160E	MAX 3000A	全デバイス
		MAX 7000S	全デバイス
APEX 20KC	EP20K200E		
ACEX®	全デバイス		

7.2.2 Quartus II Web Edition の入手とインストール

ここでは，アルテラ社のホームページから，Quartus II Web Edition 3.0 をダウンロードしてインストールするまでの実際の手順を説明します．図 7.22 に，インストールの流れを示します．

図 7.23 に示すアルテラ社のホームページから，「製品情報」→「デザイン・ソフトウェア」→

「Quartus II Web Edition」と選択していくと，図7.24に示す画面が現れます．

図7.24の画面で，「Quartus II Web Edition ソフトウェアをダウンロード」部分をクリックし

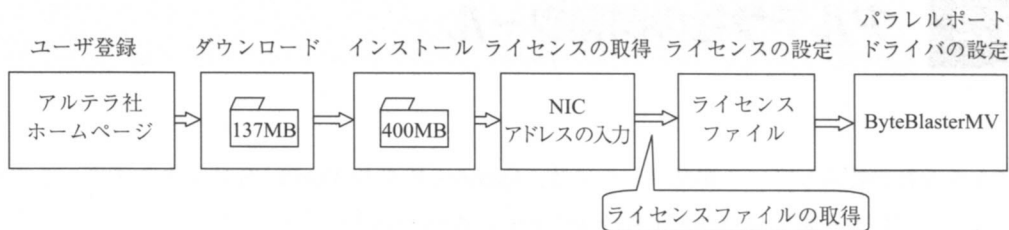



図7.22　Quartus II Web Edition 3.0 インストールの流れ

図7.23　アルテラ社のホームページ（http://www.altera.co.jp/index.html）

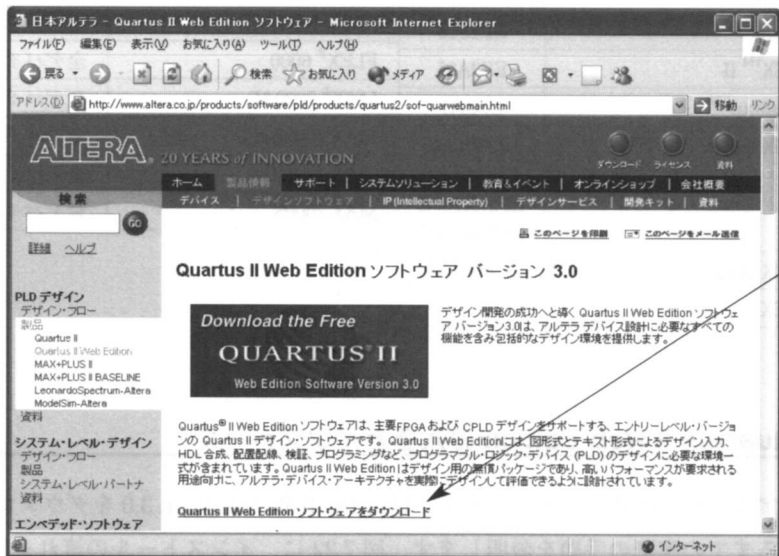



図7.24　Quartus II Web Edition 3.0 のページ

て，図 7.25 に示す画面に進みます．

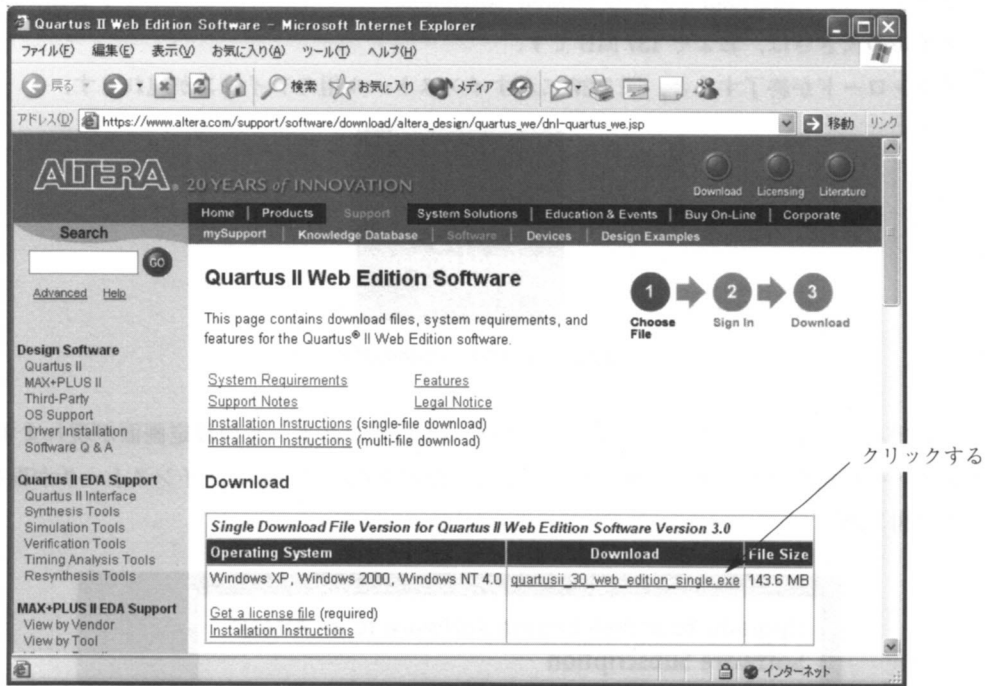
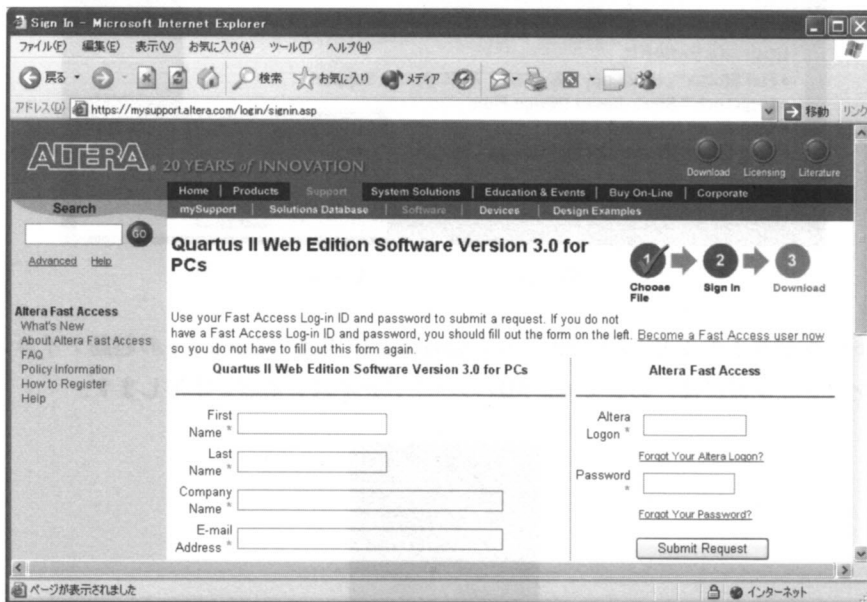



図 7.25 ダウンロードのページ

図 7.25 で，「quartusii_30_web_edition_single.exe」の部分をクリックすると，図 7.26 のユーザ登録画面が現れます．

図 7.26 ユーザ登録画面

第7章

新規登録の場合は，左側の入力欄に氏名や住所などを入力した後「Submit Request」ボタンをクリックして，「quartusii_30_web_edition_single.exe」ファイルのダウンロードを開始します．ファイルの大きさは，およそ137 MBです．

ダウンロードが終了すると，図7.27に示すインストール用のアイコンが現れます．

図7.27　インストール用のアイコン

このアイコンをダブルクリックすると，いくつかのインストールの設定画面が現れますので，インストール先のディレクトリやフォルダなどの指定を進めていくと，インストールが開始されます（図7.28）．

図7.28　インストール実行中

インストールが終了すると，図7.29に示すQuartus II Web Edition 3.0の起動アイコンが現れます．インストール後には，およそ400 MBのハードディスク領域を使用します．

図7.29　Quartus II Web Edition 3.0のアイコン

▶7.2.3 ライセンスの取得

これまでの手順で，Quartus II Web Edition 3.0 のインストールは終了しているのですが，このソフトウェアを使用するにあたっては，ライセンスの取得を行う必要があります．

ライセンスの取得を行う際には，パソコンの NIC（ネットワークインタフェースカード）のアドレスを入力する必要があります．NIC アドレスは，つぎの操作で知ることができます．

Windows XP の「スタート」ボタン→「すべてのプログラム」→「アクセサリ」→「コマンドプロンプト」を選択します（図 7.30）．

図 7.30 コマンドプロンプトを起動

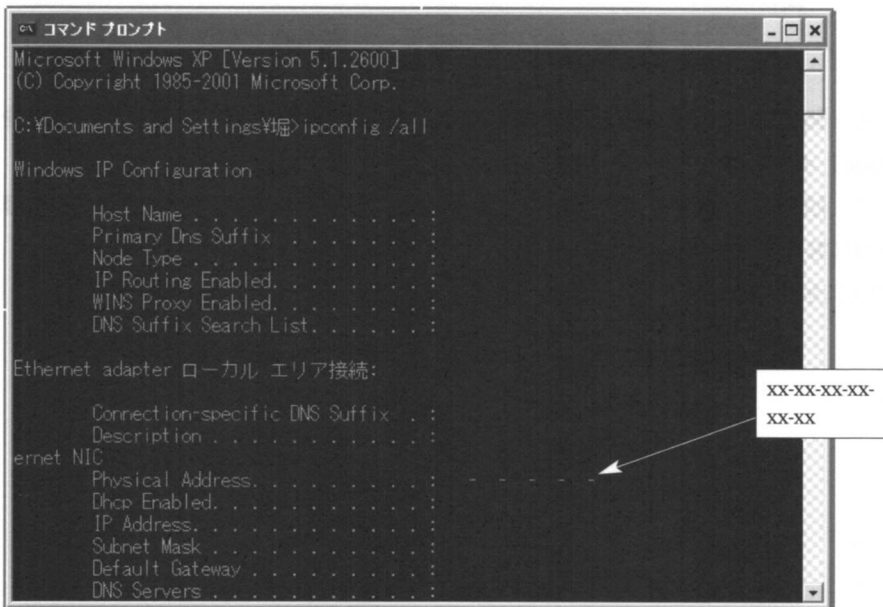



図 7.31 NIC アドレスを調べる

起動したコマンドプロンプトの画面に，「ipconfig/all」と入力し，リターンキーを押すと，パソコンのネットワークに関する各種情報が表示されます（図7.31）．表示画面を見て，2桁×6個のNICアドレスをメモしておきましょう．図7.31では，「ernet NIC Physical Address」として表示されています．番号をメモしたら，ウインドウ右上の ☒ ボタンをクリックして，コマンドプロンプトウインドウを閉じておきましょう．

つぎに，ライセンスを取得する手順について説明します．

図7.32に示す，アルテラ社のダウンロードのページ（図7.25と同じ）において，「Get a license file」の部分をクリックします．

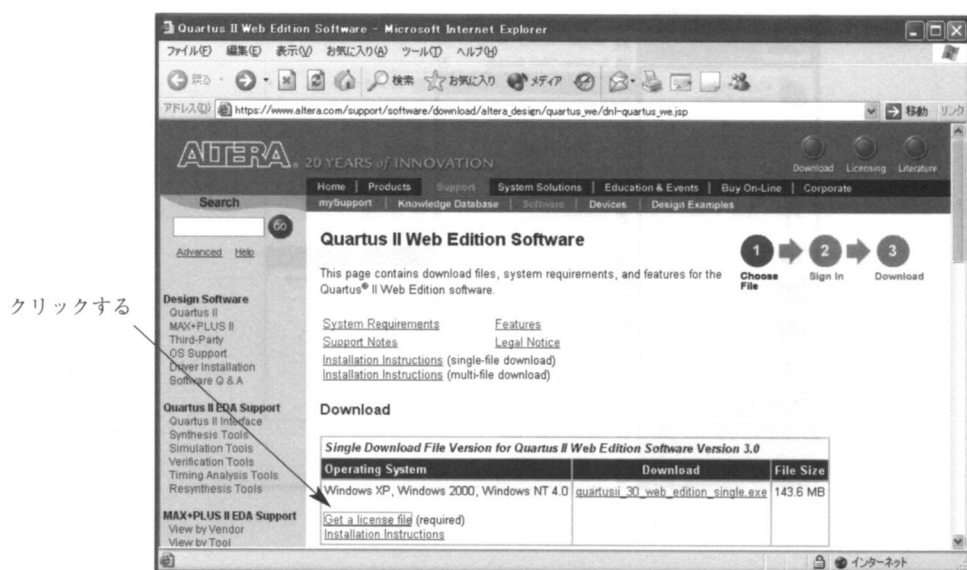



図7.32　ダウンロードのページ（図7.25と同じ）

すると，図7.33に示す画面が現れますので，先ほど調べたNICアドレスの番号を入力して「Continue」ボタンをクリックします．

続いて，図7.34に示す入力画面が現れますので，電子メールアドレスや氏名，住所などすべての項目に答えて「Continue」ボタンをクリックします．

図7.35に示す画面でも，「Continue」ボタンをクリックして先に進みます．

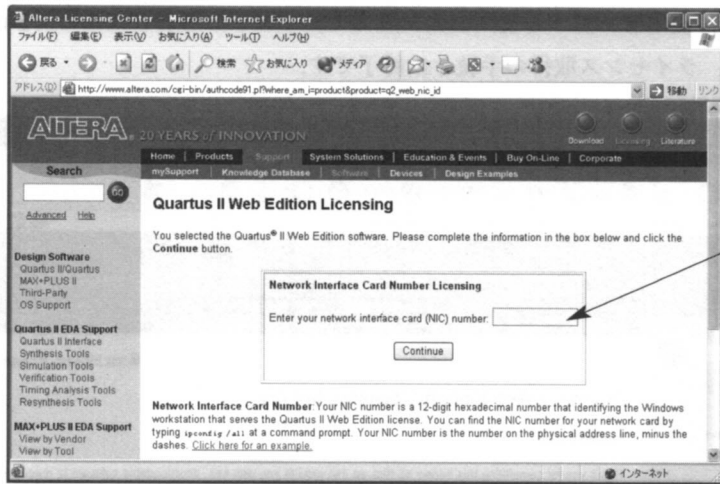



図 7.33　NIC アドレス番号の入力

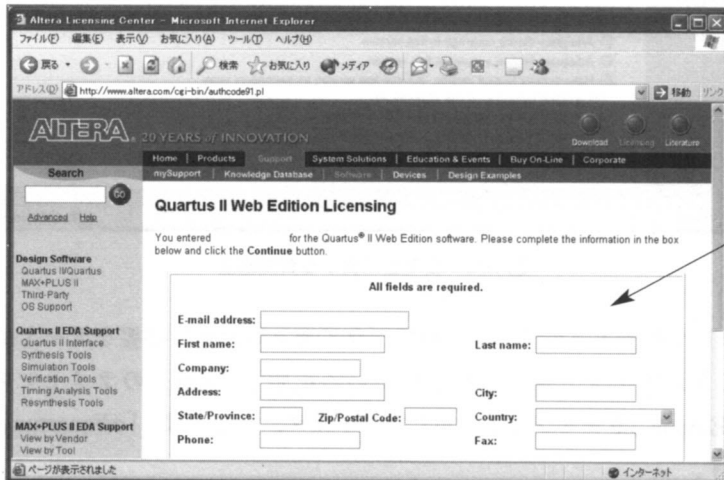



図 7.34　ライセンス情報の入力

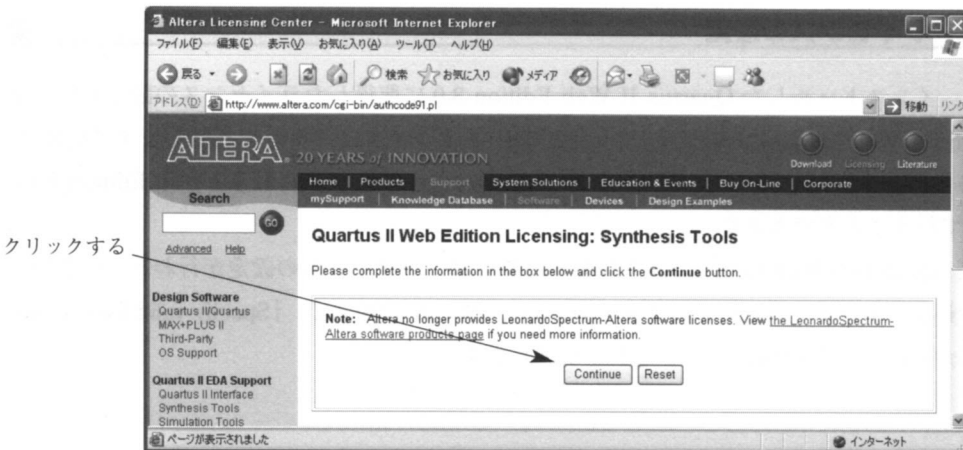



図 7.35　先へ進む「Continue」ボタンをクリック

図7.36に示す，使用者に関する質問画面に答えて，ページの終わりにある「Finish」ボタンをクリックすれば，ライセンス取得の手続きは終了です．

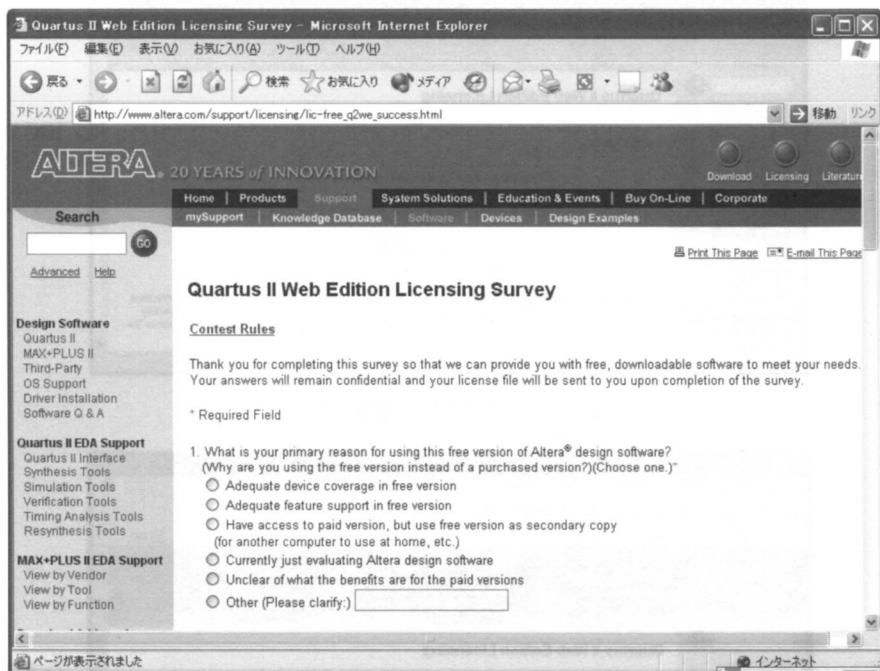



図7.36　使用者に関する質問画面

図7.34で入力した電子メールアドレス宛に，ライセンス設定に必要なデータが送信されてきます．送られてきた電子メールに添付されている拡張子「dat」のライセンスファイルは，Quartus II Web Edition 3.0をインストールしたフォルダ，たとえば，「c:¥quartus」フォルダにコピーしておきます．

▶7.2.4　ライセンスの設定

つぎに，インストールしたQuartus II Web Edition 3.0に取得したライセンスの設定を行います．Quartus II Web Edition 3.0のアイコン（図7.29）をダブルクリックして起動します．起動は，「スタート」ボタン→「すべてのプログラム」→「Altera」→「Quartus II 3.0 Web Edition Full」と選択して行うこともできます．

初めて，Quartus II Web Edition 3.0を起動する際には，ライセンスの設定が行われていないことを警告するウインドウが表示されます（図7.37）．図7.37の画面で，「Specify valid lisence file」欄にチェックを入れて「OK」ボタンをクリックします．

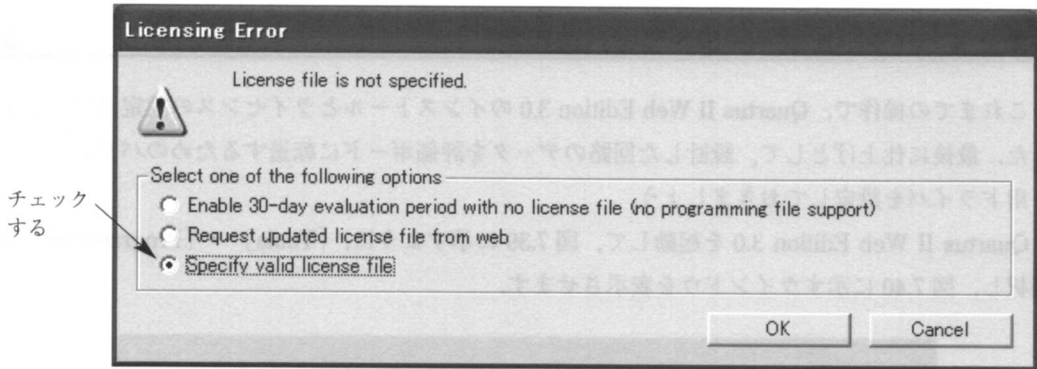



図 7.37 ライセンスの設定を警告するウインドウ

続いて，図 7.38 に示すライセンスの設定画面が現れますので，先ほどアルテラ社より送信されてきた拡張子「dat」のファイルを指定します．たとえば，「c:¥quartus」フォルダ内のライセンスファイルを指定します．

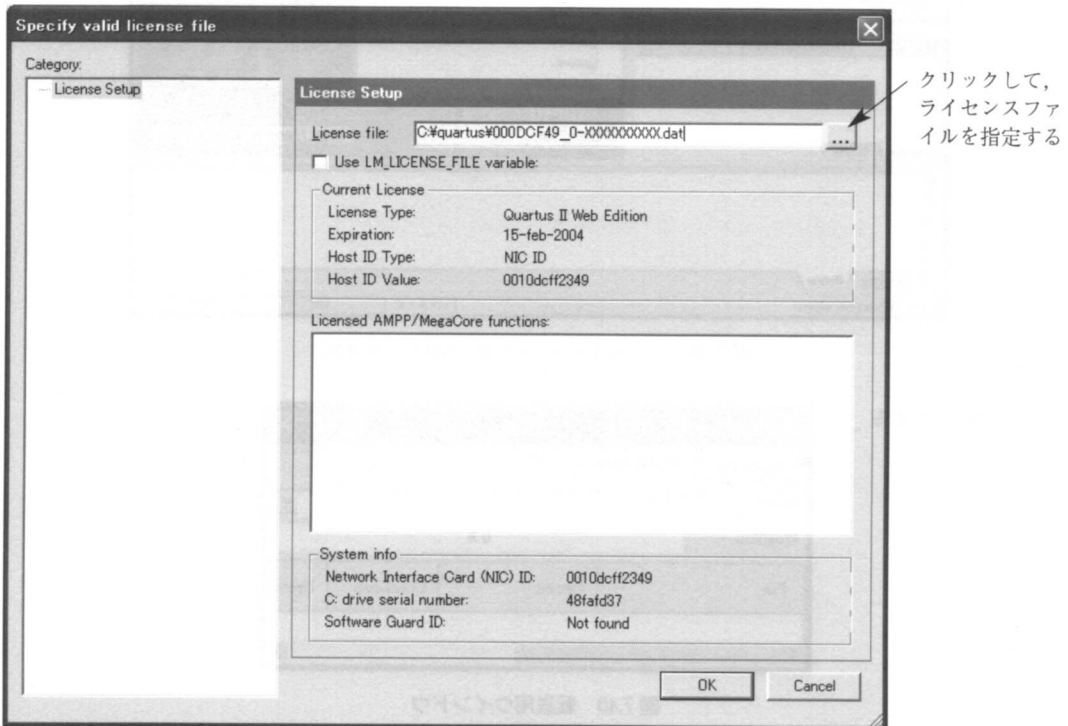



図 7.38 ライセンスファイルの指定

ライセンスファイルの指定をして「OK」ボタンをクリックすれば，Quartus II Web Edition 3.0 のライセンスの設定は終了です．

▶7.2.5 パラレルポート用ドライバの設定

これまでの操作で，Quartus II Web Edition 3.0のインストールとライセンスの設定は終了しました．最後に仕上げとして，設計した回路のデータを評価ボードに転送するためのパラレルポート用ドライバを設定しておきましょう．

Quartus II Web Edition 3.0を起動して，図7.39に示すように，「Tools」→「Programmer」を選択し，図7.40に示すウインドウを表示させます．

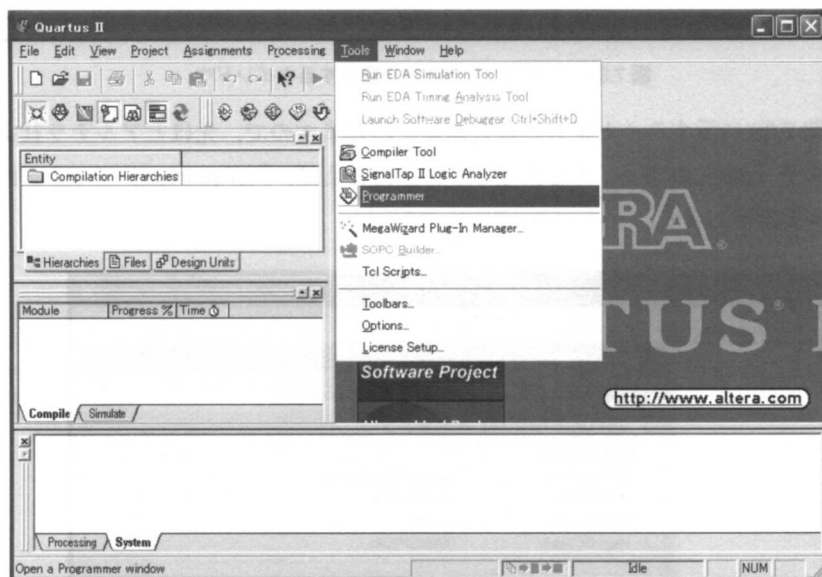



図7.39　「Tools」→「Programmer」を選択

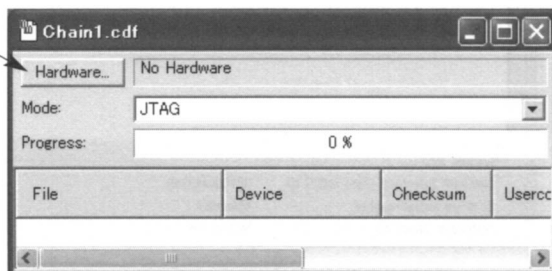



図7.40　転送用ウインドウ

「Hardware」ボタンをクリックして表示される「Hardware Setup」ウインドウにおいて，「Add Hardware」ボタンをクリックします（図7.41）．

図7.42に示すウインドウで，「ByteBlasterMV or ByteBlasterII」，および「LPT1」を選択して「OK」ボタンをクリックします．

「Hardware」メニューの中に表示される「ByteBlasterMV」を選択して，「Select Hardware」ボタンと「Close」ボタンをクリックします（図7.43）．

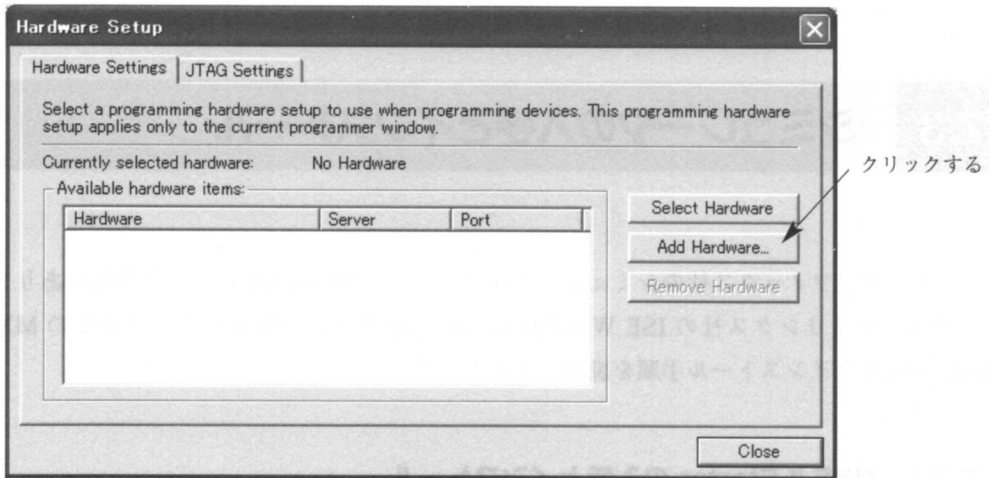



図 7.41 「Hardware Setup」ウインドウ

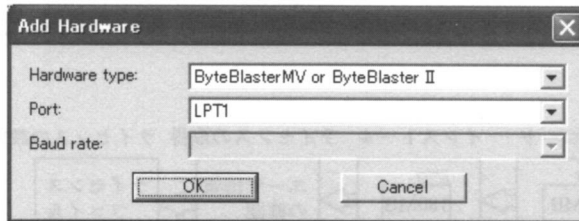



図 7.42 「Add Hardware」ウインドウ

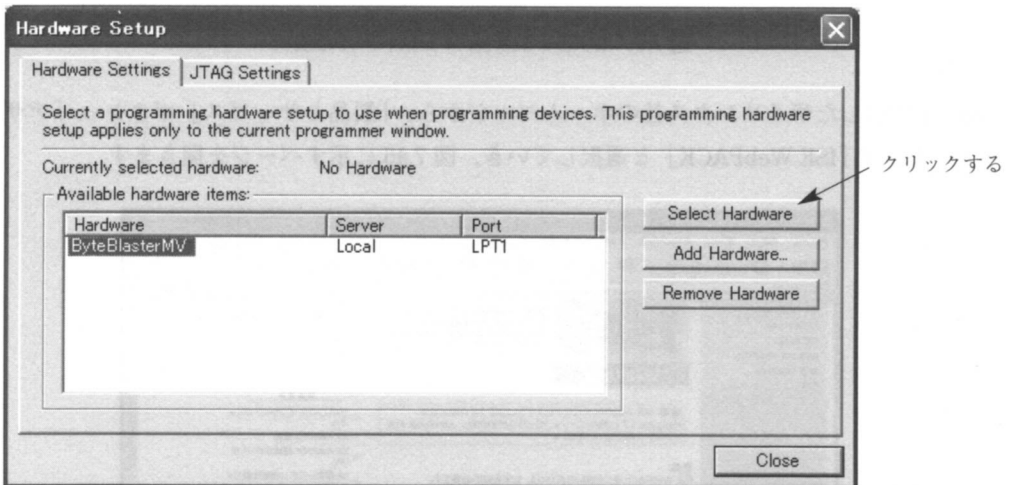



図 7.43 「ByteBlasterVM」を選択

以上で設定は終了です．Quartus II Web Edition 3.0 を用いて設計した回路データをパラレルポート（プリンタポート）経由で評価ボードに転送することが可能になりました．

7.3 シミュレータの入手とインストール

メンターグラフィックス社のシミュレータソフトウェアModelSimには，無償版があります．ここでは，ザイリンクス社のISE WebPACK 6.1iに組み込んで使用できる無償版のMXE II Starterの入手とインストール手順を説明します．

7.3.1 MXE II Starterの入手とインストール

ここでは，MXE II Starterをザイリンクス社のホームページからダウンロードして，Windows XPパソコンにインストールするまでの実際の手順を説明します．図7.44に，インストールの流れを示します．

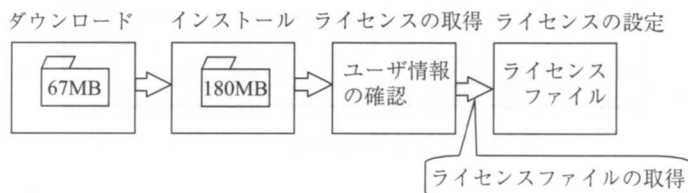



図7.44　MXE II Starterインストールの流れ

図7.2に示したザイリンクス社のホームページから，「製品とサービス」ボタン→「デザインリソース」→「ISE WebPACK」と選択していき，図7.45に示すページを開きます．

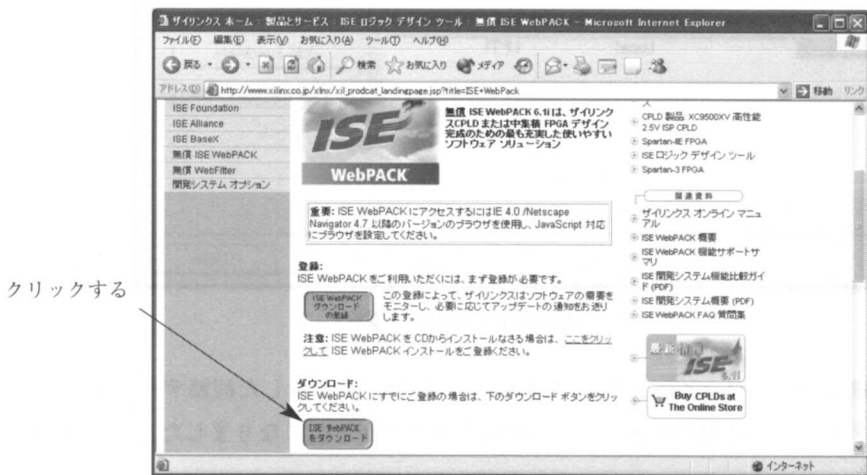



図7.45　「ISE WebPACKをダウンロード」ボタンをクリック

すでにユーザ登録を済ませている方は，ここで，「ISE WebPACK をダウンロード」ボタンをクリックした後，ユーザ ID とパスワードを入力します．

続いて表示される画面の終わり方にある「完全な MXE シミュレータ」の部分をクリックします（図 7.46）．

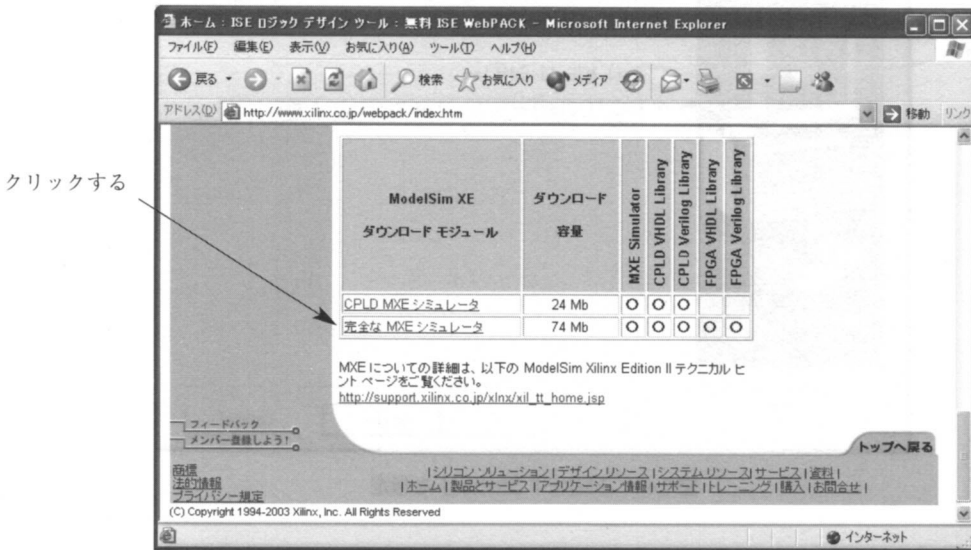



図 7.46 「完全な MXE シミュレータ」の部分をクリック

適当な保存先フォルダなどを指定すると，インストーラファイル「MXE_5.7c_Full_installer.exe」のダウンロードが始まります．ダウンロードするファイルの大きさは，およそ 67 MB です．インストールが終了すると，図 7.47 に示すインストーラのアイコンが現れますのでダブルクリックして起動します．

図 7.47 ModelSim インストーラのアイコン

図 7.48 に示す確認用ウインドウの「OK」ボタンをクリックします．

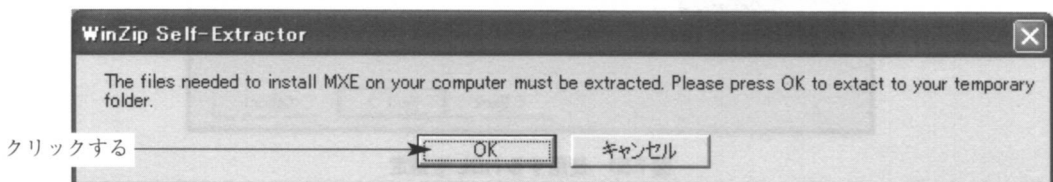



図 7.48 インストール確認用ウインドウ

図7.49に示すウインドウで，「MXE II Starter－Limited Version of MXE II (Free」をチェックして「Next」ボタンをクリックします．

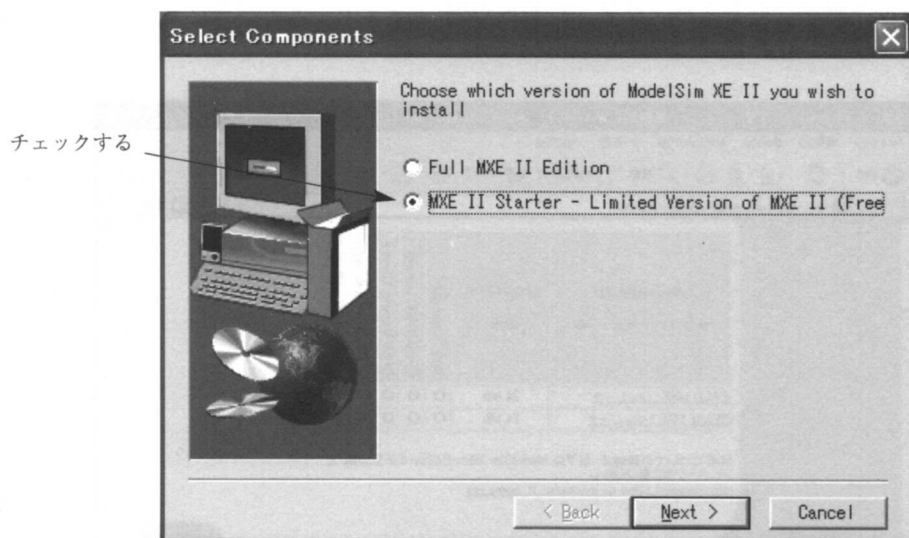



図7.49　「MXE II Starter」を指定

引き続いて，ライセンスの同意やインストール先フォルダの設定ウインドウなどが現れますので，「Next」ボタンなどをクリックして進んでいきます．

図7.50に示すウインドウでは，使用するHDLの種類を設定します．MXE II Starterでは，インストール時にHDLを指定しなければなりません．ここでは，「Full VHDL」を指定して「Next」ボタンをクリックします．

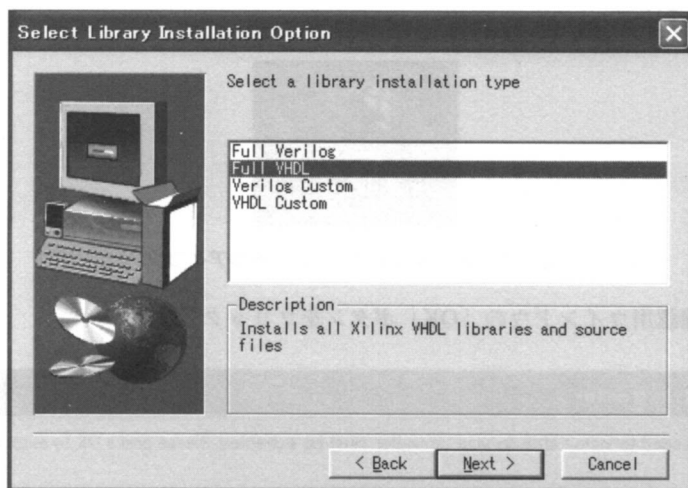



図7.50　使用するHDLを指定

続いて，いくつかの確認ウインドウが表示されますので，メッセージを確認して先へ進みましょう．図7.51に示すウインドウの「Finish」ボタンをクリックすれば，インストールは終了で

す．インストール後には，およそ180 MBのハードディスク領域を必要とします．

図7.51　インストールの最終ウインドウ

ディスクトップにアイコン作成を指定していれば，図7.52に示すショートカット用アイコンが表示されているはずです．

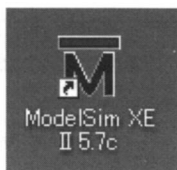



図7.52　MXE II Starterのアイコン

▶7.3.2　ライセンスの取得

MXE II Starterを使用する場合には，ライセンスを取得しておく必要があります．ここでは，インターネットを利用してライセンスを取得する手順を説明します．手続きの際には，使用するパソコンのハードウェア情報が自動的に送信されますので，MXE II Starterをインストールしたパソコンを使用して操作を行ってください．

これまでの操作で，MXE II Starterのインストールを終了すれば，図7.51に示したウインドウに続いて，図7.53に示すウインドウが表示されます．

すでに，ザイリンクス社へユーザ登録を済ませている方は，「Continue」の部分をクリックした後，ユーザIDとパスワードを入力して先へ進みます．

図7.54に示すユーザ情報ウインドウが表示されたら，内容を確認して「Submit」ボタンをクリックします．

そして，続いて表示される確認用ウインドウに答えれば，登録した電子メールアドレス宛にライセンス情報が送信されます．

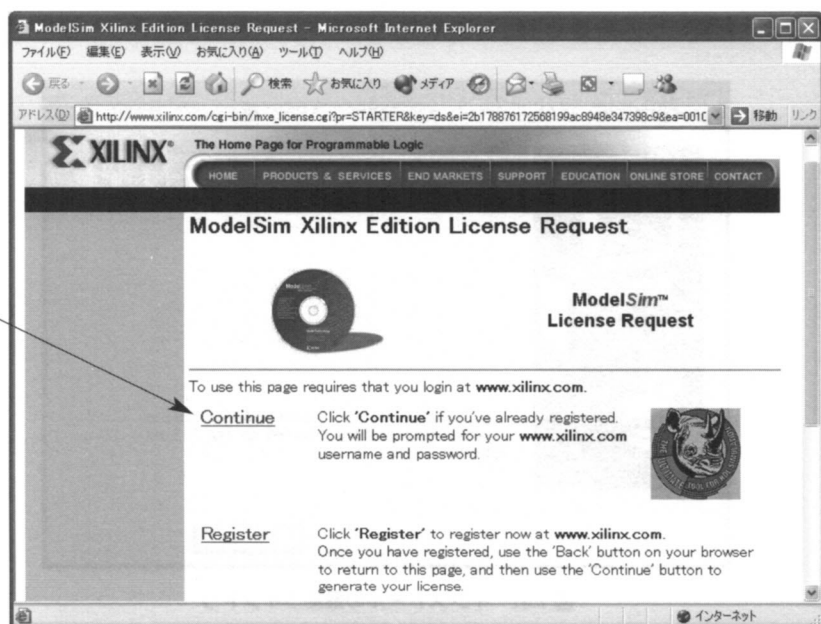



図 7.53　ライセンスの取得ウインドウ

図 7.54　ユーザ情報ウインドウ

▶7.3.3　ライセンスの設定

ライセンスの取得によって送られてきた電子メールには，拡張子「dat」のライセンスファイルが添付されています．ここでは，このライセンスファイルを用いてライセンスの設定を行う手順を説明します．ライセンスファイルは，設定によって MXE II Starter のフォルダにコピーされますので，とりあえずはディスクトップ上においておきます（図 7.55）．

図 7.55 送信されてきたライセンスファイル

Windows XP の「スタート」ボタン→「すべてのプログラム」→「ModelSim XE II 5.7c」→「Licensing Wizard」を選択してライセンスウィザードを起動します（図 7.56）.

図 7.56 ライセンスウィザードの起動

図 7.57 に示すウインドウが表示されますので，「Continue」ボタンをクリックします.

図 7.57 ライセンスウィザードの起動画面

図7.58に示すウインドウで，ライセンスファイル（図7.55）の場所を指定します．この際，ディレクトリ名に全角文字を使用するとうまくいかない場合もあるようです．エラーメッセージの出る場合には，たとえば「C:¥」などにライセンスファイルをおいておけばよいでしょう．

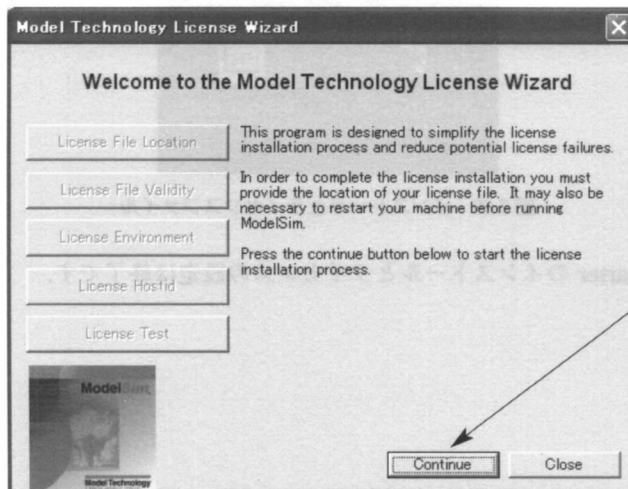
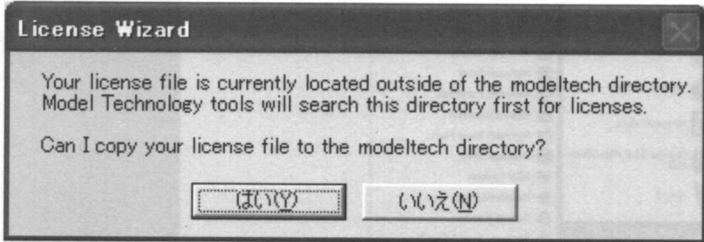
図7.58　ライセンスファイルの場所を指定

続いていくつかのメッセージウインドウ（図7.59など）が表示されますので，内容を確認して先へ進めばライセンスの設定は終了します．

図7.59　メッセージウインドウ

設定終了後は，ディスクトップ上においたライセンスファイルやMXE II Starterインストーラファイル（図7.60）を削除しておきましょう．

図7.60　インストーラとライセンスファイル

以上で，MXE II Starterのインストールとライセンスの設定は終了です．

7.4 評価ボードの概要

本書では，ソリトンウェーブ社の評価ボード「HDL トレーナー」を例にして実習の手順を説明しています．「HDL トレーナー」は，低価格な評価ボードですが，ザイリンクス社のCoolRunnerII（XC2C256）を搭載しており，実習には十分な性能をもった製品です．ここでは，「HDL トレーナー」の概要について理解しましょう．

異なる評価ボードを使用する場合でも，スイッチや LED などの入出力装置と CPLD のピン割り当てなどを考慮すれば，ほぼ同様の実習を行えることでしょう．

7.4.1 HDL トレーナーの概要

図 7.61 に，HDL トレーナー（HDL－10）一式の外観を示します．

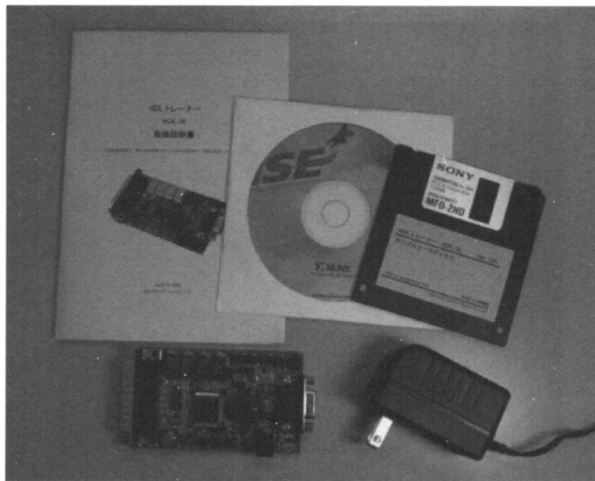



図 7.61 HDL トレーナー一式の外観

HDL トレーナーには，つぎに示すものが含まれています．

- HDL トレーナー本体基板
- AC アダプタ
- 取り扱い説明書
- ザイリンクス社 ISE WebPACK インストール用 CD-ROM
- サンプルプログラムディスク
- ネジ 4 本

付属しているネジを本体基板のパラレルコネクタ（D-SUBコネクタ）に付いているナットと交換すると，本体基板をパソコンのプリンタポートへ直接接続できるようになります（図7.62）．もちろん，D-SUB 25ピンの延長ケーブル（オスーメス）を別途用意すれば，パソコンの前面に本体基板をおいて実習を行うことも可能です（図7.63）．

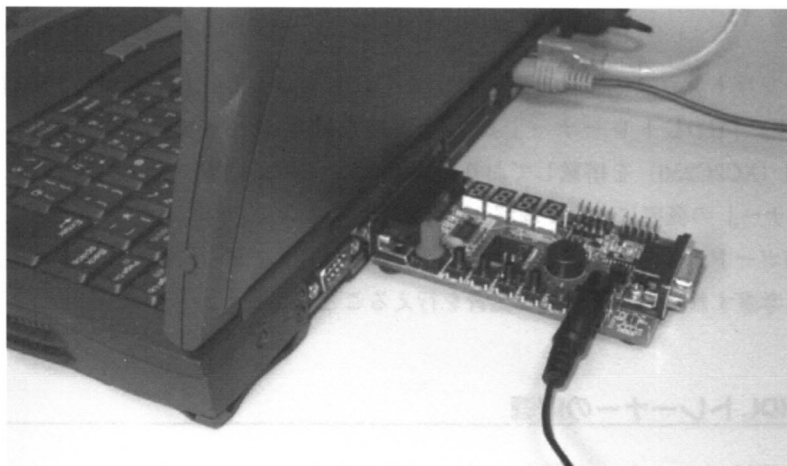



図7.62　パソコンに直接接続した例

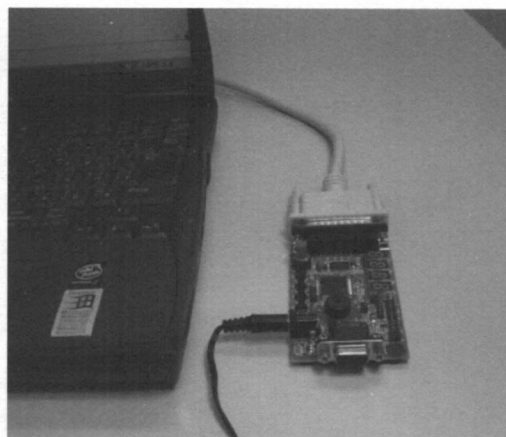



図7.63　延長ケーブルで接続した例

HDLトレーナーの特徴を以下にまとめます．

- 実習には十分な256マクロセルのCPLDを搭載
- 入力装置として，プッシュスイッチ5個，16進ディップスイッチ1個を搭載
- 出力装置として，7セグメントLED4個，ブザー1個を搭載
- JTAGインタフェース搭載
- パラレルコネクタ，シリアル通信用コネクタ各1個を搭載
- 簡易型A-D，D-Aコンバータ搭載
- 拡張用入出力端子を搭載

・保証期間 1 年間

・このクラスでは，群を抜く低価格（キャンペーン価格 9800 円）

図 7.64 に，本体基板の各部を示します．

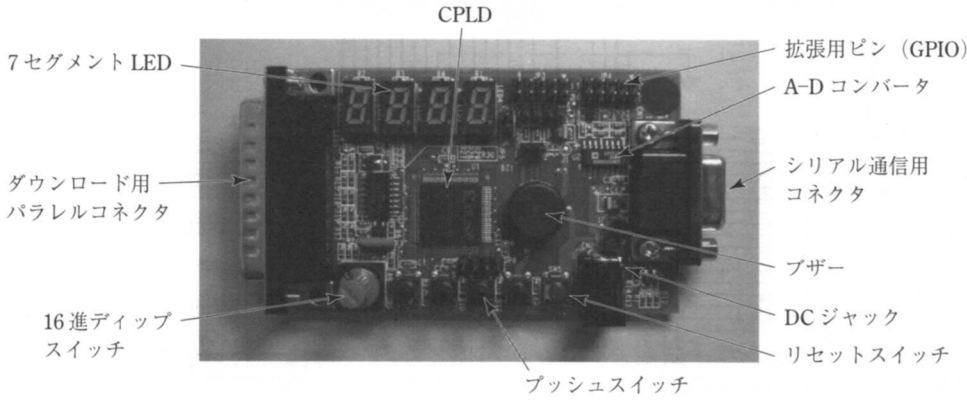



図 7.64 HDL トレーナーの各部

HDL トレーナーに搭載されている CPLD のピンは，表 7.3 のように割り付けられています．

表 7.3 CPLD のピン割り当て

No.	接続先	I/O	No.	接続先	I/O	No.	接続先	I/O	No.	接続先	I/O
1	DEB0	INOUT	26	VCC1.8V		51	VCC3.3V		76	LEDB7	OUT
2	DEB1	INOUT	27	EXT	OUT	52	LEDD6	OUT	77	LEDB6	OUT
3	DEB2	INOUT	28	SPKOUT	OUT	53	LEDD5	OUT	78	LEDB5	OUT
4	DEB3	INOUT	29	HEXSW3	IN	54	NC		79	LEDB4	OUT
5	VAUX	IN	30	HEXSW2	IN	55	LEDD4	OUT	80	LEDB3	OUT
6	DEB4	INOUT	31	GND		56	LEDD3	OUT	81	LEDB2	OUT
7	DEB5	INOUT	32	HEXSW1	IN	57	VCC1.8V		82	LEDB1	OUT
8	DEB6	INOUT	33	HEXSW0	IN	58	LEDD2	OUT	83	TDO	OUT
9	DEB7	INOUT	34	SW1	IN	59	NC		84	GND	
10	DEB8	INOUT	35	SW2	IN	60	LEDD1	OUT	85	LEDB0	OUT
11	DEB9	INOUT	36	SW3	IN	61	LEDD0	OUT	86	LEDA7	OUT
12	DEB10	INOUT	37	SW4	IN	62	GND		87	LEDA6	OUT
13	DEB11	INOUT	38	VCC3.3V		63	NC		88	VCC3.3V	
14	DEB12	INOUT	39	AIN	INOUT	64	LEDC7	OUT	89	LEDA5	OUT
15	DEB13	INOUT	40	BZOUT	OUT	65	NC		90	LEDA4	OUT
16	DEB14	INOUT	41	IRXD	IN	66	NC		91	LEDA3	OUT
17	DEB15	INOUT	42	ITXD	OUT	67	LEDC6	OUT	92	LEDA2	OUT
18	NC		43	ICTS	OUT	68	LEDC5	OUT	93	NC	
19	NC		44	NC		69	GND		94	LEDA1	OUT
20	VCC3.3V		45	TDI	IN	70	LEDC4	OUT	95	NC	
21	GND		46	NC		71	LEDC3	OUT	96	NC	
22	CLK1	IN	47	TMS	IN	72	LEDC2	OUT	97	LEDA0	OUT
23	CLK2	INOUT	48	TCK	IN	73	LEDC1	OUT	98	VCC3.3V	
24	CLK2	INOUT	49	IRTS	IN	74	LEDC0	OUT	99	nRESET	IN
25	GND		50	LEDD7	OUT	75	GND		100	GND	

※ SW1 ～ 4，nRESET，LEDA ～ LEDD は負論理の信号となっています．

図 7.65（a）（b）に，HDL トレーナーの全回路図を示します．

CPLD

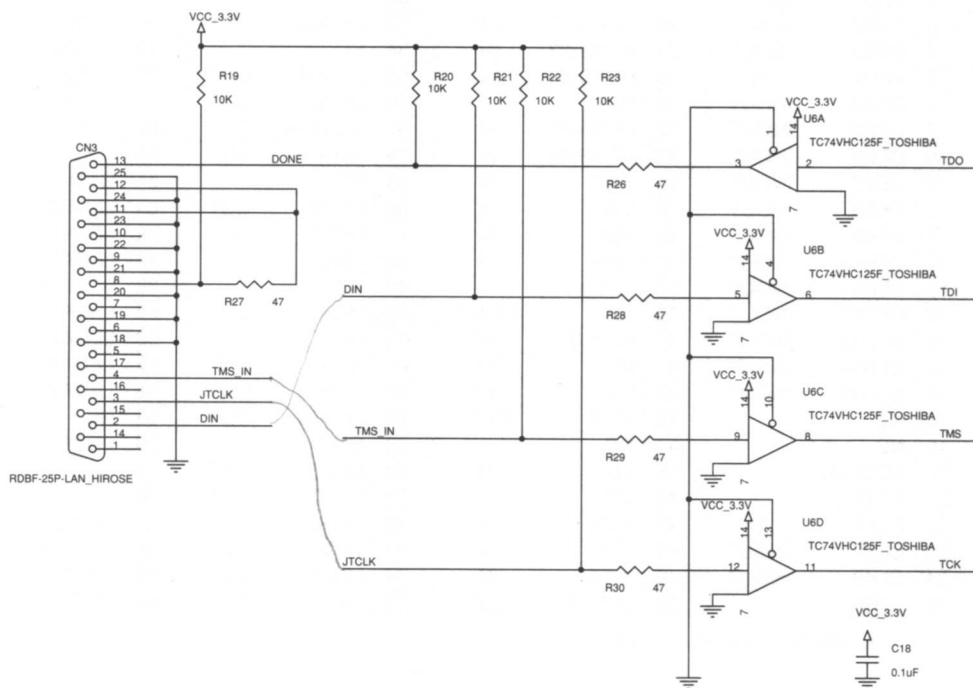



図 7.65(a)　HDL トレーナーの全回路図

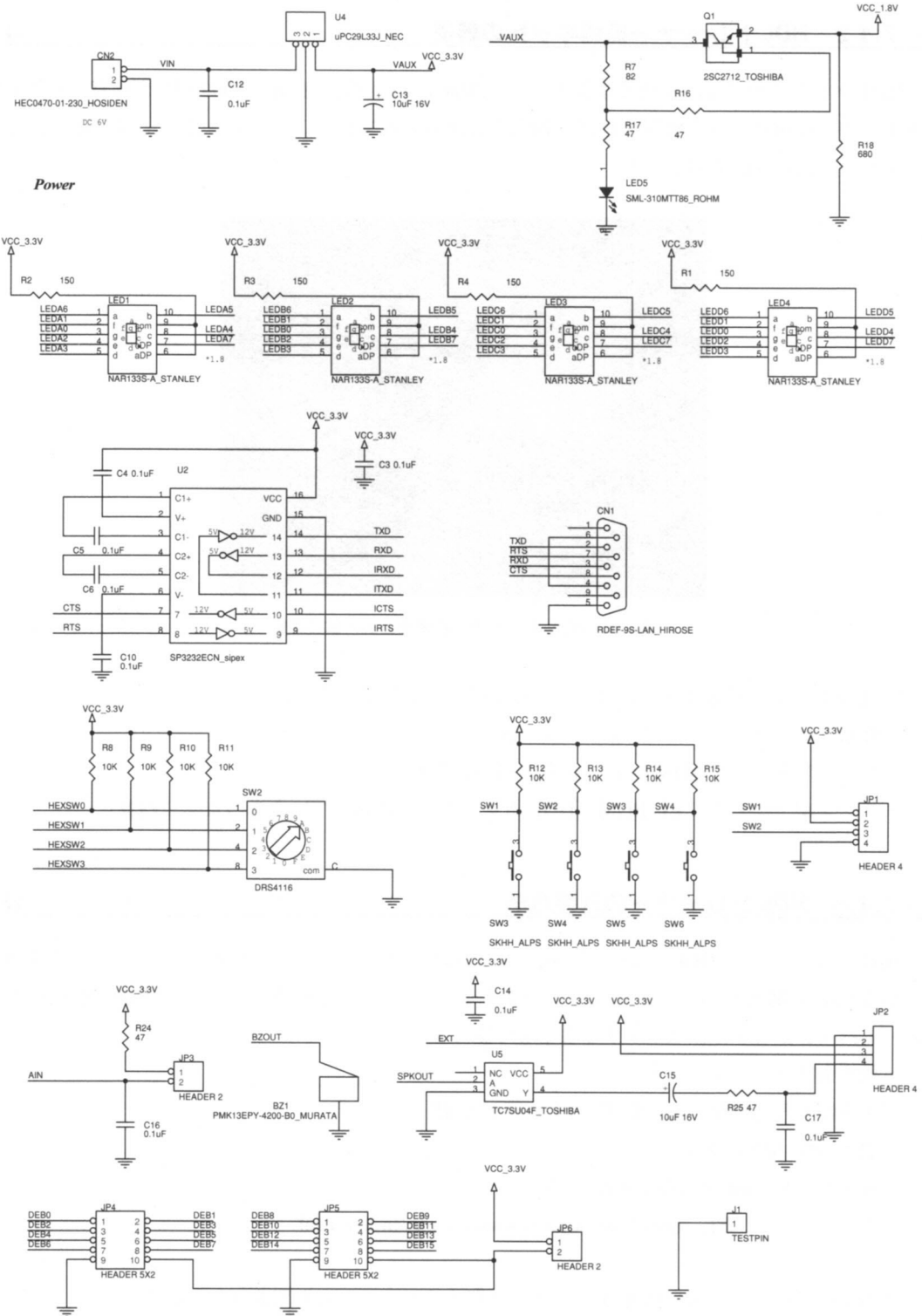



図 7.65(b) HDL トレーナーの全回路図

第7章

7.4.2 HDLトレーナー拡張キットの構成

HDLトレーナーには，別売で拡張キット（2004.2現在の販売価格は2500円）が用意されています．この拡張キットを利用すれば，幅広く各種の実習を行うことができます．図7.66に，拡張キット一式の外観を示します．

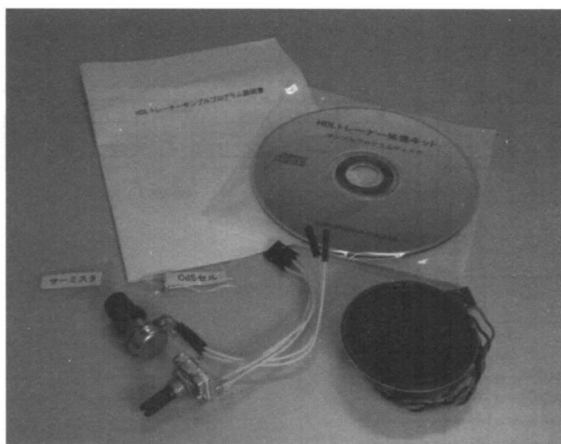



図7.66 HDLトレーナー拡張キット一式の外観

HDLトレーナー拡張キットには，つぎのものが含まれています．

・外部スピーカ　・ボリューム　・ロータリエンコーダ
・サーミスタ　・CDSセル　・取り扱い説明書
・サンプルプログラムCD-ROM（HDLトレーナーに付属しているものと同じ内容）

7.4.3 HDLトレーナーの入手方法

HDLトレーナーとHDLトレーナー拡張キットは，ソリトンウェーブ社のホームページから通信販売による購入申し込みを行うことができます．また，HDLトレーナーのPDF説明書は，ホームページからダウンロードすることも可能です．

・株式会社ソリトンウェーブ
　東京都千代田区神田松永町17-15 大野ビル4F
　TEL：03-5256-0955
　電子メール　info@solitonwave.co.jp
　ホームページURL　http://www.solitonwave.co.jp/index.html

このホームページには，HDLトレーナーに関するフォーラムも開設されています．

付録 Verilog-HDL について

Verilog-HDL は，VHDL より 8 年遅い 1995 年に IEEE によって標準化されたハードウェア記述言語ですが，VHDL と同様，多くのユーザに支持されています．どちらの言語を使用した場合でも，開発の流れは同じです．ここでは，VHDL と比較した Verilog-HDL の概要についてまとめておきます．

付.1 Verilog-HDL の書き方

例として，2 入力 AND 回路を取り上げます．VHDL によるコードをリスト付 1，Verilog-HDL によるコードをリスト付 2 に示します．

```
library IEEE;                                  ← ライブラリ宣言
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity sample1 is                              ← エンティティ宣言
    Port ( A : in std_logic;
           B : in std_logic;
           F : out std_logic);
end sample1;

architecture Behavioral of sample1 is          ← アーキテクチャ宣言

begin
     F <= A and B ;
end Behavioral;
```

リスト付 1　VHDL のコード例

```
module sample2(A,B,F);                         ← モジュール宣言
  input A;                                     ← ポート宣言，ネット宣言，レジスタ宣言
  input B;
  output F;

      assign F = A & B ;                       ← 構造・動作記述部

endmodule
```

リスト付 2　Verilog-HDL のコード例

VHDLはライブラリ宣言，エンティティ宣言，アーキテクチャ宣言から構成されますが，Verilog-HDLの基本的なコードはモジュール宣言，ポート宣言，構造・動作記述部から構成されます．つぎに，Verilog-HDLの各部について簡単に説明します．

① **モジュール宣言**：設計する回路名とポートリストを記述します．

module　[回路名]　(ポートリスト)；

② **ポート宣言**：入力と出力の宣言を記述します．

input　[入力ポート名]　；
output　[出力ポート名]　；

③ **構造・動作記述部**：設計する回路の構造や動作を記述します．assign文は，代入処理を行う文です．

assign　[代入先]　=　[代入元]　；

Verilog-HDLを使用する場合には，VHDLと異なる以下の項目について注意しましょう．

① 大文字と小文字を区別します．
② 異なるビット幅の信号でも代入処理が可能です（右詰で代入処理されます）．
③ 比較演算子の等しいには「==」を使用するなど，多くの演算子がVHDLと異なります．

付.2　Verilog-HDLの文法

付表1に，VHDLとVerilog-HDLの主な文法を比較した一覧を示します．

付表1　VHDLとVerilog-HDLの文法比較表

項目	VHDL	Verilog-HDL
コメント	--　1行のコメント	//　1行のコメント /*　*/　複数行のコメント
文字	大文字と小文字を区別しない (ただし' ', " " 内では区別する)	大文字と小文字を区別する
データ型	bit，std_logic， std_logic_vector	bit，integer character
定数	'0'，'1'， "00101111"	1'b0，1'b1 8'h2f
ファイル指定	libray IEEE； use [ファイル名]；	include " [ファイル名] "
入出力宣言	[ポート名]：[モード型] [データ型]； A：in　std_logic； B：out std_logic_vector (3 dounto 0)；	[モード型] [ポート名]； input A； output [3：0] B；

付録

項目	VHDL	Verilog-HDL
内部信号宣言	signal 信号名 : データ型 ; signal A : std_logic ; signal B : std_logic_vector（3 dounto 0）;	wire 信号名 ; wire A ; wire [3 : 0] B ;
信号代入	代入先 <= 代入元 ; （:=）変数代入	assign 代入先 = 代入元 ; （<=）コンカレントな （ノンブロッキング）代入
順次処理文	process（ ） begin 〜 end process ;	always @（ ） begin 〜 end
トリガ	CLK' event and CLK ='1' CLK' event and CLK ='0'	posedge CLK negedge CLK
if	if（ 条件式 ）then 処理 1 ; else 処理 2 ; end if ;	if（ 条件式 ） 処理 1 ; else 処理 2 ;
	if（ 条件式 1 ）then 処理 1 ; elsif（ 条件式 2 ）then 処理 2 ; else 処理 3 ; end if ;	if（ 条件式 1 ） 処理 1 ; else if（ 条件式 2 ） 処理 2 ; else 処理 3 ;
case	case 信号名 is when（ 信号の値 1 => 処理 1 ; when（ 信号の値 2 => 処理 2 ; when others => 処理 n ; end case ;	case（信号名） 信号の値 1 : 処理 1 ; 信号の値 2 : 処理 2 ; default : 処理 n ; end case

付録

項目	VHDL		Verilog-HDL	
演算子	and, or, not, xor, nand, nor	(論理)	&&, ||, !	(論理)
			&, |, ~, ^, ~&, ~|,	(ビット)
	=, /=, >, <, >=, <=	(関係)	==, !=, >, <, >=, <=	(関係)
	+, −, *, /, mod	(算術)	+, −, *, /, %	(算術)
	信号名 & 信号名	(連接)	[信号名 , 信号名]	(連接)
階層設計	component コンポーネント名 port (ポート名 : モード型 データ型); end component ; ラベル : コンポーネント名 port map (接続);		module 上位のモジュール名 (ポートリスト); ポート宣言 下位のモジュール名 ラベル (接続); end module	

付.3　Verilog-HDLのコード例

代表例として，10秒カウンタ（138ページ例題5−1参照）のVerilog-HDLコードをリスト付3～6に示します．139ページのリスト5.6～5.9と比較してみるとよいでしょう．また，このコードでは，階層設計を行っていますので，この点からもVHDLとVerilog-HDLを比べてみましょう．

ISE WebPACKを使用してVerilog-HDLのコードを記述する場合には，ツールバーの「Project」→「New Source」を選択した際に，「Verilog Module」を指定します（39ページの図2.15，図2.16参照）．

```
module watch10(CLK,RESET,DOUT);
  input CLK;
  input RESET;
  output [ 7:0]  DOUT;
  wire CE_TMP;
  wire [ 3:0] CNT_TMP;

divider22 C1(.CLK(CLK),.CEOUT(CE_TMP));
counter10 C2(.CE(CE_TMP),.RESET(RESET),.CNT(CNT_TMP));
decoder8 C3(.DIN(CNT_TMP),.DOUT(DOUT));

endmodule
```

リスト付 3　階層設計用（watch10）のコード

```
module divider22(CLK,CEOUT);
  input CLK;
  output CEOUT;
  reg CEOUT;
  reg [ 21:0] DD;

always @(posedge CLK)
begin
 DD <= DD+1'b1;
end

always @(DD)
begin
        if (DD == 22'b1111_1111_1111_1111_1111_11)
                 CEOUT <= 1'b1;
        else
                 CEOUT <= 1'b0;
end
```

リスト付 4　分周回路（divider2）のコード

```
module counter10(CE,RESET,CNT);
  input CE;
  input RESET;
  output [ 3:0]  CNT;
  reg  [ 3:0]  CNT;

always @(posedge CE)
begin
        if(RESET == 1'b0)
                CNT <= 4'b0000;
        else if(CNT == 4'b1001)
                         CNT <= 4'b0000;
                else
                         CNT <= CNT +1'b1;
end
endmodule
```

リスト付 5　カウンタ（counter10）のコード

付録

```
module decoder8(DIN,DOUT);
    input [ 3:0]  DIN;
    output [ 7:0]  DOUT;
    reg  [ 7:0]  DOUT;

always @(DIN)
begin
    case(DIN)
       4'b0000:DOUT <= 8'b0000_0011 ;
       4'b0001:DOUT <= 8'b1001_1111 ;
       4'b0010:DOUT <= 8'b0010_0101 ;
       4'b0011:DOUT <= 8'b0000_1101 ;
       4'b0100:DOUT <= 8'b1001_1001 ;
       4'b0101:DOUT <= 8'b0100_1001 ;
       4'b0110:DOUT <= 8'b0100_0001 ;
       4'b0111:DOUT <= 8'b0001_1011 ;
       4'b1000:DOUT <= 8'b0000_0001 ;
       4'b1001:DOUT <= 8'b0000_1001 ;
       default:DOUT <= 8'b1111_1111 ;
    endcase
    end
endmodule
```

リスト付 6　デコーダ（decoder8）のコード

142 ページの表 5.3 に示したピン割り当てを行って HDL トレーナーで動作確認を行ってください．ただし，Verilog-HDL を使用してシミュレーションを行う場合には，MXE II Starter を Verilog-HDL 用にインストールしてあることが必要です．

付録

演習問題解答

第1章

1.1　CPUに与えるプログラムはCPUというハードウェア上での動作を示す命令であるが，CPLD/FPGAに与えるコードはCPLD/FPGA内部を目的のハードウェアに仕立てる指示である．

1.2　①，④，⑤

1.3　CPLD　①，②，④
FPGA　③，⑤

1.4　VHDL

1.5　論理シミュレーションは実際の回路で生じる遅延の影響を考慮せずに行うが，遅延シミュレーションはそれを考慮する．

1.6　シミュレーションを行う場合に，回路に与える入力信号を記述したデータ．

1.7　HDLで記述したソースコードをディジタル回路に変換する．

1.8　論理合成によって生成された回路を，実際のCPLD/FPGA内部でどのように構成するかを決め，ピン配置を行う．

1.9　作成したデータをCPLD/FPGAへ転送する．

1.10　ダウンロードする際の規格はJTAGまたはUSBなどを使用することができる．JTAGの場合はパソコンのパラレルポート（プリンタポート），USBの場合にはUSBポートを使用する．

1.11　①，④，⑤，⑦，⑧
JTAGを使用する場合において「ダウンロードケーブル」といった場合には，データ変換装置（電子回路）を含んだケーブルを指す．一方で，評価ボードの中にはこのデータ変換装置を搭載したものがある．このような評価ボードを用いる場合には，パソコンとは単純な（データ変換回路を含まない）ケーブルによって接続すればよい．

第2章

2.1　規模，入出力ピン数，動作電圧，動作速度，パッケージ形状，価格など

2.2　CoolRunner XPLA3ファミリの「XCR3128XL-VQ100C」が条件を満たす最小規模のCPLDとなる．

2.3　高速で動作させるほど，消費電力は大きくなる．

2.4　① 10進数で0から9までをカウントする4ビットのカウンタ回路である．
② 評価ボードに搭載されている4 MHzの動作クロックを分周する22ビットのカウンタ回路である．分周回路の出力信号によって，10進数カウンタ回路が，およそ1秒の周期で動作するようにしている．
③ 10進数カウンタ回路の出力は2進数の0000から1001であるため，この出力データで7セグメントLEDを10進数表示できるようにデータ変換する回路である．

2.5　*CLK*: 外部からCPLDへ入力される1ビットのクロック信号（4 MHz）である．
RESET: 外部からCPLDへ入力される1ビットのリセット信号（プッシュスイッチ）である．
LED: CPLDから外部（7セグメントLED）へ出力される8ビット（0～7）の表示信号である．

2.6　VHDLのコードが記述されたテキストファイルである．

2.7　56ページ図2.46において，ファイル「LED10.npl」のアイコンをダブルクリックすればよい．

2.8　① 配置配線から作業を行う．たとえば，51ページ図2.37において，ファイル「LED10.ucf」の内容を変更すればよい．その後，配置配線を実行し，ダウンロードを行う．

②

表 2.8　変更後の入出力ピン

記号		ピン番号	備考
入力	*CLK*	22	セラロック 4 MHz
	RESET	35	プッシュスイッチ SW2
出力	*LED0*	74	7 セグメント LED LED3
	LED1	73	
	LED2	72	
	LED3	71	
	LED4	70	
	LED5	68	
	LED6	67	
	LED7	64	

③ 各自で実習を行うこと．

第 3 章

3.1 ① 1 ビットの論理データ型　② 複数ビットの論理データ型，つまり「std_logic」のベクトル型

3.2 シーケンシャル文は実行が順次行われるが，コンカレント文では同時に行われる．

3.3 センシティビティリストに記述した信号に変化があった場合，process 文中に記述した処理が実行される．

3.4 構造記述は回路図に基づいた記述を行うが，動作記述は真理値表などで表された動作を直接記述することができる．

3.5 ① in std_logic　② in std_logic　③ in std_logic　④ out std_logic
⑤ signal　⑥ begin　⑦ not　⑧ X or Y

3.6 case 文を用いたコード例をリスト 3.30 に示す．

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity syo3_6 is
   Port ( A : in std_logic_vector(2 downto 0);
          F : out std_logic);
end syo3_6;

architecture Behavioral of syo3_6 is

begin
        process(A)
        begin
                case (A) is
                         when "000" => F <= '1' ;
                         when "111" => F <= '1' ;
                         when others => F <= '0' ;
                end case ;
        end process ;
end Behavioral;
```

リスト 3.30　3 ビット一致回路のコード（動作記述）

第 4 章

4.1 ① J & K　② CLK　③ '0'　④ RESET　⑤ '0'　⑥ case　⑦ not WORK
⑧ others　⑨ end case　⑩ WORK

4.2

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity syo4_2 is
   Port ( CLK,RESET : in std_logic;
           Q : out std_logic_vector(1 downto 0));
end syo4_2;

architecture Behavioral of syo4_2 is
        signal WORK : std_logic_vector(1 downto 0);
begin
process(CLK)
begin
   if(CLK'event and CLK='1') then
                if (RESET = '0') then
                        WORK <= "00" ;
                else
                        WORK <= WORK + '1' ;
                end if ;
   end if;
end process;
Q <= WORK ;
end Behavioral;
```

リスト 4.23　同期式 4 進カウンタのコード

4.3

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity syo4_3 is
   Port ( CLK,RESET : in std_logic;
           Q : out std_logic_vector(3 downto 0));
end syo4_3;

architecture Behavioral of syo4_3 is
        signal WORK : std_logic_vector(3 downto 0);
begin
process(CLK)
begin
   if(CLK'event and CLK='0') then
                if ((RESET = '0') or (WORK = "1000")) then
                         WORK <= "0000";
                else
                         WORK <= WORK + '1' ;
           end if;
        end if;
end process;
Q <= WORK ;
end Behavioral;
```

リスト 4.24　同期式 9 進カウンタのコード

4.4

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity syo4_4 is
   Port (CLK : in std_logic;
         LED : out std_logic_vector(7 downto 0));
end syo4_4;

architecture Behavioral of syo4_4 is
        signal DD : std_logic_vector(21 downto 0) ;
        signal CE : std_logic ;
        signal WORK : std_logic_vector(3 downto 0) ;
begin

-- 分周回路　２２ビット
process (CLK)
begin
  if CLK'event and CLK='1' then
      DD <= DD + '1';
  end if;
end process;

process (DD)
begin
  if DD = "1111111111111111111111" then
     CE <= '1';
  else
     CE <= '0';
  end if;
end process;

-- カウンタ回路
process(CE)
begin
        if (CE'event and CE='0') then
                  if(WORK = "0111") then
                          WORK <= "0000";
                  else
                          WORK <= WORK - '1' ;
                  end if ;
        end if;
end process;

--デコーダ回路
process(WORK)
begin
        case WORK is
               when "0000" => LED <= "00000011" ;  --0
               when "1111" => LED <= "00001001" ;  --9
               when "1110" => LED <= "00000001" ;  --8
               when "1101" => LED <= "00011011" ;  --7
               when "1100" => LED <= "01000001" ;  --6
               when "1011" => LED <= "01001001" ;  --5
               when "1010" => LED <= "10011001" ;  --4
```

解答

```
                when "1001" => LED <= "00001101" ;  --3
                when "1000" => LED <= "00100101" ;  --2
                when "0111" => LED <= "10011111" ;  --1
                when others => null ;
        end case;
end process;
end Behavioral;
```

リスト 4.25　10 進ダウンカウンタ表示回路のコード

第 5 章

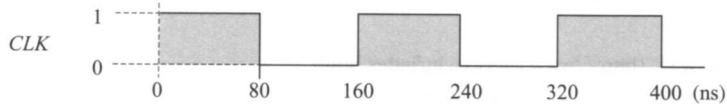
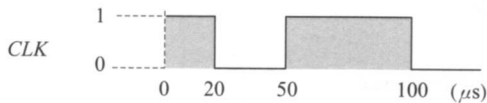
5.1　① entity　② component　③ end component　④ HA2_CO　⑤ ha　⑥ A　⑦ S
⑧ HA1_CO　⑨ C_I　⑩ or　⑪ entity　⑫ ha　⑬ ha　⑭ xor　⑮ and

5.2　複雑な回路でも，モジュール単位に分割して考えることができる．また，何度も使用する機能は，モジュールとして 1 度記述しておけばよい．

5.3　130 ページ図 5.5，図 5.6 参照

第 6 章

6.1　②，③

6.2　リスト 6.7

リスト 6.8

6.3　① '0'　② process　③ '1'　④ 10　⑤ end process

6.4　152 ページのリスト 6.2 と同様なテストベンチを作成して，VHDL で記述した半加算器（79 ページ例題 3－1 参照）をシミュレーションする．

6.5　162 ページのリスト 6.6 と同様なテストベンチを作成して，VHDL で記述した同期式 5 進カウンタ（120 ページリスト 4.16 参照）をシミュレーションする．

解答

参考文献

1. ザイリンクス社の資料（http://www.xilinx.co.jp/）
- CoolRunner-II CPLD ファミリ
- XC2C256 : 256 マクロセル CoolRunner-II CPLD
- CPLD/CoolRunner 製品概要
- Spartan 製品概要
- 6.1i ソフトウェア　マニュアル　など多数

2. メメックジャパン社の資料（http://www.memec.co.jp/）
- ISE6.1iJ インストールマニュアル
- ModelSim インストールガイド
- 2003 Xilinx and Memec Product Guid
- トレーニング資料（VHDL 初級・中級コース）

3. アルテラ社の資料（http://www.altera.co.jp/index.html）
- デザイン・ソフトウェア・セレクタ・ガイド
- Quartus II version 3.0 Software SP2 Release Notes
- Quartus II Software Quick Start Guide
- Introduction to Quartus II Manual
- Quartus II Installation & Licensing for PCs Manual

4. ソリトンウェーブ社の資料
- HDL トレーナー（HDL－10）取扱説明書

5. VHDL によるハードウェア設計入門　長谷川裕恭　CQ 出版社
6. VHDL によるディジタル回路入門　並木秀明，永井亘道　技術評論社
7. VHDL，Verilog，AHDL によるデジタルシステム設計　原田豊　丸善
8. 入門 Verilog-HDL 記述　小林優　CQ 出版社
9. ディジタル電子回路の基礎　堀桂太郎　東京電機大学出版局

索　引

［和　文］

あ　行

か　行

さ　行

た　行

な　行

は　行

著　者　略　歴

堀　桂太郎（ほり・けいたろう）

日本大学大学院　理工学研究科

博士後期課程　情報科学専攻修了

博士（工学）

現在　国立明石工業高等専門学校

　　　電気情報工学科　助教授

<主な著書>

図解 PIC マイコン実習（森北出版）

H8 マイコン入門（東京電機大学出版局）

ディジタル電子回路の基礎（東京電機大学出版局）

アナログ電子回路の基礎（東京電機大学出版局）

絵ときディジタル回路入門早わかり（オーム社）など

図解 VHDL 実習 —ゼロからわかるハードウェア記述言語—

2004 年 4 月 15 日　第 1 版第 1 刷発行　

著　　者　堀　桂太郎

発 行 者　森北　肇

発 行 所　**森北出版株式会社**

東京都千代田区富士見 1－4－11（〒 102－0071）

電話 03－3265－8341 ／ FAX 03－3264－8709

http://www.morikita.co.jp/

日本書籍出版協会・自然科学書協会・工学書協会　会員

JCLS ＜(株) 日本著作出版権管理システム委託出版物＞

落丁・乱丁本はお取替えいたします　　印刷/双文社印刷所・製本/協栄製本

Printed in Japan ／ ISBN4-627-78391-4

図解 PIC マイコン実習

ゼロからわかる電子制御

堀　桂太郎／著

B5 判・208 頁・定価 2310 円⑤

初めて PIC を学習しようとする方々を対象に，PIC の中でも特に扱いやすい 16F84A を選び，その原理と使い方を例題や演習問題をまじえてやさしく解説.

■目次　マイコン制御の基礎／PIC マイコンの基礎／マイコンでのデータ表現／アセンブラ言語／プログラミング実習／MPLAB の使い方／付録

[ISBN4-627-78331-0]